VIOLATION DETECTION, EXTRA INFORMATION RELEASE AND SECURE IMPLEMENTATION FOR DIFFERENTIALLY PRIVATE MECHANISMS

A Dissertation in

Computer Science and Engineering

by

Ding Ding

© 2022 Ding Ding

Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

August 2022
The dissertation of Ding Ding was reviewed and approved by the following:

Daniel Kifer  
Professor of Computer Science and Engineering  
Dissertation Co-Adviser  
Co-Chair of Committee

Danfeng Zhang  
Associate Professor of Computer Science and Engineering  
Dissertation Co-Adviser  
Co-Chair of Committee

Gang Tan  
Professor of Computer Science and Engineering

Aleksandra Slavković  
Professor of Statistics

Chita R. Das  
Professor of Computer Science and Engineering  
Department Head of Computer Science and Engineering
Abstract

While the accelerating growth of data has led to fruitful research and real-world applications, numerous reported incidents of data leakage and abuse have raised public concerns for individual privacy. Among many attempts to protect the confidentiality of users who provide data, differential privacy has become the gold standard for extracting information from sensitive data. It provides strong privacy protection against powerful adversaries by ensuring that any individual’s data has very little influence on the released outcome.

Although differential privacy has seen huge success both in academia and in industry, it is facing several challenges in practice. First, there is no free lunch in data privacy. Asking for a little more privacy will usually come at a cost, often to the accuracy of some analysis. The accuracy of private data releases is very important in many applications, and much research has been devoted to achieving better privacy/utility trade-offs. Second, due to the subtle nature of this privacy definition, significant errors have appeared even in peer-reviewed papers and systems. Moreover, almost all current implementations of differential privacy suffer from a type of side channel vulnerability that results from finite precision and rounding effects of floating-point operations.

In this dissertation, we propose the first ever counterexample generator in the literature that is capable of detecting violations of differential privacy by producing counterexamples for incorrect algorithms. The counterexamples are designed to be short and human-understandable so that the counterexample generator can be used in the development process – a developer could quickly explore variations of an algorithm and investigate where they break down. Our approach is statistical in nature. It runs a candidate algorithm many times and uses statistical tests to try to detect violations of differential privacy.

We also investigate an important class of algorithms called private selection which make choices among outcomes based on a confidential dataset. They serve as key components in many privacy preserving algorithms. Through careful analysis of their privacy proofs, we discover that many of the private selection algorithms can be made strictly more powerful by releasing additional information
at no extra cost to privacy. This additional information can boost the confidence in selected outcomes or significantly increase the accuracy of subsequent analysis. Furthermore, we show that many of our improved private selection algorithms can be implemented securely so that they are immune to attacks that exploit the floating-point vulnerability.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................................. vii

LIST OF TABLES ................................................................................................................... viii

Acknowledgments .................................................................................................................... ix

Chapter 1 Introduction ............................................................................................................ 1

Chapter 2 Related Work .......................................................................................................... 5

Chapter 3 Background ............................................................................................................. 8

  3.1 Differential Privacy .................................................................................................... 8
  3.2 Global Sensitivity and the Laplace Mechanism ............................................................. 9

Chapter 4 Violation Detection for Privacy Mechanisms ............................................................. 11

  4.1 Introduction ............................................................................................................... 11
  4.2 Background ............................................................................................................... 13
    4.2.1 Hypothesis Testing ........................................................................................... 13
  4.3 Counterexample Detection .......................................................................................... 14
    4.3.1 Overview ......................................................................................................... 16
    4.3.2 Hypothesis Testing ........................................................................................... 19
    4.3.3 Event Selection ................................................................................................ 21
    4.3.4 Input Generation .............................................................................................. 24
  4.4 Experiments .............................................................................................................. 28
    4.4.1 Noisy Max ....................................................................................................... 29
    4.4.2 Histogram ........................................................................................................ 32
    4.4.3 Sparse Vector ................................................................................................... 33
    4.4.4 Performance .................................................................................................... 37

Chapter 5 Extra Information Release from Private Selection Mechanisms ............................ 41

  5.1 Introduction ............................................................................................................... 41
  5.2 Background and Notation ........................................................................................... 44
  5.3 Randomness Alignment .............................................................................................. 45
  5.4 Improving Sparse Vector ............................................................................................ 49
    5.4.1 Adaptive SVT with Gap .................................................................................... 49
    5.4.2 Using Exponential or Geometric Noise ............................................................. 54
    5.4.3 Utilizing Gap Information ................................................................................. 57
LIST OF FIGURES

Figure 4-1  Interpreting experimental results on hypothesis tests. A hypothetical algorithm $M$ claims to achieve $\epsilon_0$-differential privacy. For each $\epsilon$ from 0.2 to 2.4 we would evaluate if $M$ satisfies $\epsilon$-differential privacy. We show a typical graph when $M$ does satisfy $\epsilon_0$-differential privacy (left), a graph where $M$ possibly provides more privacy (center) and a graph where $M$ provides less privacy than advertised. ................................................................. 17

Figure 4-2  Results of Noisy Max algorithm and its variants. .......................................................... 29

Figure 4-3  Results of Histogram algorithm and its variants ............................................................. 33

Figure 4-4  Results for variants of Sparse Vector Technique .......................................................... 39

Figure 5-1  The blue dots are values of $\theta_{\min} = \text{argmin}(\frac{e^{\theta \epsilon}}{[e^{\theta \epsilon} - 1]^2} + \frac{e^{(1-\theta) \epsilon/k}}{(e^{(1-\theta) \epsilon/k} - 1)^2})$ for $k$ from 1 to 50. The orange curve is the function $\theta = 1/(1 + \sqrt{k^2})$. ................................................................. 57

Figure 5-2  Percent reduction of Mean Squared Error on monotonic queries, for different $k$, for SVT with Gap and Noisy Top-K with Gap when half the privacy budget is used for query selection and the other half is used for measurement of their answers. Privacy budget $\epsilon = 0.7$. .......................................................... 73

Figure 5-3  Percent reduction of Mean Squared Error on monotonic queries, for different $\epsilon$, for SVT with Gap and Noisy Top-K with Gap when half the privacy budget is used for query selection and the other half is used for measurement of their answers. The value of $k$ is set to 10. .......................................................... 74

Figure 5-4  # of queries answered by SVT and Adaptive SVT with Gap under different $k$’s for monotonic queries. Privacy budget $\epsilon = 0.7$ and x-axis: $k$. .......................................................... 75

Figure 5-5  Precision and F-Measure of SVT and Adaptive SVT with Gap under different $k$’s for monotonic queries. Privacy budget $\epsilon = 0.7$ and x-axis: $k$. .......................................................... 75

Figure 5-6  Remaining privacy budget when Adaptive SVT with Gap is stopped after answering $k$ queries using different datasets. Privacy budget $\epsilon = 0.7$. .......................................................... 77

Figure 5-7  Percentage of remaining privacy budget of hybrid algorithms, Noisy Top-K and SVT on the BMS-POS dataset. Results on Kosarak and T40I10D100K are similar. Privacy budget $\epsilon = 0.7$. .......................................................... 78

Figure 5-8  Average query answers of the hybrid algorithms, Noisy Top-K and SVT on the BMS-POS dataset. Results on Kosarak and T40I10D100K are similar. Privacy budget $\epsilon = 0.7$. .......................................................... 78

Figure 5-9  The estimated probability of $p \leq \alpha$ when the output index from Exponential Mechanism with Gap is not optimal. Utility scores are sampled from BMS-POS ........................................ 79
LIST OF TABLES

Table 4-1  Database Categories and Samples .......................................................... 25
Table 4-2  Counterexamples Detected for Incorrect Privacy Mechanisms .................. 40
Table 4-3  Running Time for Testing Privacy Mechanisms ...................................... 40
Table 5-1  Commonly Used Noise Distributions ......................................................... 45
Table 5-2  Notation for Randomness Alignment ....................................................... 45
Table 5-3  Statistics of Datasets .............................................................................. 73
Table 6-1  Notation ................................................................................................. 95
Table 6-2  Noise Distributions ............................................................................... 95
Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisors Dr. Daniel Kifer and Dr. Danfeng Zhang for having introduced me to the field of privacy and for their guidance, encouragement and support throughout my studies at Penn State University. Their enthusiasm in this field, along with their extensive knowledge and innovative ideas is what drove me forward over the past years, to which I am forever grateful. I can only hope to be as inspiring and insightful a mentor and researcher as they are. I have benefited enormously from the excellent seminars organized by Dr. Aleksandra Slavković. I wish to thank Dr. Gang Tan and Dr. Aleksandra Slavković for serving on my defense committee. Their helpful feedback and comments have vastly improved the quality of my research and this dissertation.

During my adventures as a Ph.D. candidate at Penn State, I have had the fortune to meet with many outstanding friends and colleagues, to whom I owe my thanks. My first project in this field is in collaboration with Yuxin Wang, who is not only an excellent researcher, but also a dear friend to me. I have enjoyed conversations and collaborations with many other amazing friends and colleagues including but certainly not limited to Yingtai Xiao, Guanhong Wang, Shen Liu, Peixuan Li, Robert Brotzman, Yongzhe Huang, Jonathan Hehir, Jeffrey Ching and Dongrui Zeng.

I would like to thank my father Qiang Ding and my mother Liping Guo for their unconditional love and unwavering support over the past years. They have been the driving force and motivation of my life.

Last, I would like to thank my beloved wife Shengzi Songwei and my children Felix & Chris Ding. They have created such a warm home for me and brought so much meaning and joy to my life. There are simply no words in this world to express my feelings for them.

Funding Acknowledgment The work presented in this dissertation is supported by NSF CNS-1228669, CNS-1702760, CNS-1816282, CNS-1566411, CNS-1931686 and a gift from Facebook. The findings and conclusions presented in this document are those of the authors and do not necessarily reflect the views of the organizations listed previously.
Dedication

To my beloved wife Shengzi Songwei.
Chapter 1

Introduction

Differential privacy [1] has become a de facto standard for extracting information from a dataset (e.g., answering queries, building machine learning models, etc.) while protecting the confidentiality of individuals whose data are collected. Implemented correctly, it guarantees that any individual’s record has very little influence on the output of the algorithm. Industry and government agencies are increasingly adopting differential privacy to protect the confidentiality of users who provide data. Current and planned major applications include data gathering by Google [2, 3], Apple [4], and Microsoft [5]; database querying by Uber [6]; and publication of population statistics at the U.S. Census Bureau [7, 8, 9, 10].

Although differential privacy has seen huge success both in academia and in industry, it is facing several challenges in practice. First, the design of differentially private algorithms is very subtle and error-prone – it is well-known that a large number of published algorithms are incorrect (i.e. they violate differential privacy). A sign of this problem is the existence of papers that are solely designed to point out errors in other papers [11, 12]. The problem is not limited to novices who may not understand the subtleties of differential privacy; it even affects experts whose goal is to design sophisticated algorithms for accurately releasing statistics about data while preserving privacy.

Second, there is no free lunch in data privacy. Asking for a little more privacy will usually come at a cost, often to the accuracy of some analysis. The accuracy of private data releases is very important in many applications, and much research has been devoted to achieving better privacy/utility trade-offs. One way to improve accuracy is to increase the value of the privacy parameter $\epsilon$, known as the privacy loss budget, as it provides a trade-off between an algorithm’s utility and its privacy protections. However, values of $\epsilon$ that are deemed too high can subject a company to criticisms of not providing enough privacy [13]. For this reason, researchers invest significant effort in tuning algorithms [14, 15, 16, 17, 18, 19] and privacy analyses [20, 21, 18, 22] to provide better utility at the same privacy cost.

Lastly, many current implementations of differential privacy suffer from a type of side channel
vulnerability called the floating-point vulnerability [23]. Most differentially private algorithms require sampling from statistical distributions. However, the sampling procedures implemented in most software libraries result in porous distributions over double-precision numbers, deviating in important respects from their mathematical abstractions. While (pure) differential privacy requires all outputs to be feasible for all possible inputs and having similar probabilities when the inputs are close, the results of floating-point arithmetic will be concentrated on a small subset of outputs. Careful examination of these subsets demonstrate that they overlap only partially on close inputs, thus breaking the guarantee of differential privacy, which would have applied if all operations were computed with infinite precision and unlimited source of entropy.

In this dissertation, we present the first ever counterexample generator in the literature that is capable of detecting violations of differential privacy by producing counterexamples for incorrect algorithms. We envision that such a counterexample generator would be useful in the development cycle – variations of an algorithm can be quickly evaluated and buggy versions could be discarded (without wasting the developer’s time in a manual search for counterexamples or a doomed search for a correctness proof). Furthermore, counterexamples can help developers understand why their algorithms fail to satisfy differential privacy and thus can help them fix the problems. This feature is absent in all existing programming platforms and verification tools. To the best of our knowledge, this is the first paper that treats the problem of detecting counterexamples in incorrect implementations of differential privacy.

We also investigate an important class of algorithms called private selection mechanisms, which make choices among outcomes based on a confidential dataset. They include the Noisy Max [24], Sparse Vector Technique (SVT) [24, 11], and the Exponential Mechanism [25]. They serve as key components in many privacy preserving algorithms for synthetic data generation, ordered statistics, frequent itemset mining, hyperparameter tuning for statistical models, etc. Through careful analysis of their privacy proofs [26, 27], we discovered that many of the private selection algorithms can be made strictly more powerful by releasing additional information at no extra cost to privacy. This additional information can boost the confidence in selected outcomes or increase the accuracy of subsequent analysis by up to 66%. Moreover, certain online algorithms that process queries on the fly can be made to adaptively choose the noise scale for each query. This adaptivity allows the algorithm to process significantly more queries with the same privacy budget (or answer the same number of queries but with plentiful leftover budget
that can be used for other purposes). Furthermore, we show that many of our improved private selection algorithms can be implemented without floating-point vulnerability.

In summary, in this dissertation we make the following contributions:

- We present the first counterexample generator for differential privacy. It treats programs as semi-black-boxes and uses statistical tests to detect violations of differential privacy.

- We provide a simplified template for writing correctness proofs for intricate differentially private algorithms.

- Using this technique, we propose and prove the correctness of two new mechanisms: Noisy Top-K with Gap and Adaptive SVT with Gap. These algorithms improve on the original versions of Noisy Max and SVT by taking advantage of free information (i.e., information that can be released at no additional privacy cost) that those algorithms inadvertently throw away. We also show that the free gap information can be maintained even when these algorithms use one-sided noise. This variation improves the accuracy of the gap information.

- We demonstrate some of the uses of the gap information that is provided by these new mechanisms. When an algorithm needs to use Noisy Max or SVT to select some queries and then measure them (i.e., obtain their noisy answers), we show how the gap information from our new mechanisms can be used to improve the accuracy of the noisy measurements. We also show how the gap information in SVT can be used to estimate the confidence that a query’s true answer really is larger than the threshold.

- We show that the Exponential Mechanism can also release free gap information. Noting that the free gap extensions of Noisy Max and SVT required access to the internal state of those algorithms, we show that this is unnecessary for Exponential Mechanism. This is useful because implementations of Exponential Mechanism can be very complex and use a variety of different sampling routines.

- We propose two novel hybridizations of Noisy Max and SVT. These algorithms can release the identities of the approximate top-$k$ queries as long as they are larger than a pre-specified threshold. If fewer than $k$ queries are returned, the algorithms save privacy budget and the gap information
they release directly turns into estimates of the query answers (i.e., the algorithm returns the query identities and their answers for free). If \( k \) queries are returned then the algorithms still return the gaps between their answers.

- We empirically evaluate the mechanisms on a variety of datasets to demonstrate their improved utility.

- We present secure implementations of Noisy Top-K with Gap that is free of the floating-point vulnerability.

**Outline** The rest of this dissertation is organized as follows. We discuss related work in Chapter 2 and relevant background on differential privacy in Chapter 3. We present our counterexample detector in Chapter 4, our proof template and improved private selection mechanisms in Chapter 5 and their secure implementations in Chapter 6. Finally, we conclude this dissertation in Chapter 7 and share our thoughts on potential directions for future work in Chapter 8.
Chapter 2

Related Work

**Differential privacy**  The term differential privacy covers a family of privacy definitions that include pure $\epsilon$-differential privacy (the topic of this paper) [1] and its relaxations: approximate $(\epsilon, \delta)$-differential privacy [28], concentrated differential privacy [29, 20], and Renyi differential privacy [21]. The pure and approximate versions have received the most attention from algorithm designers (e.g., see the book [24]). However, due to the lack of availability of easy-to-use debugging and verification tools, a considerable fraction of published algorithms are incorrect. In this paper, we focus on algorithms for which there is a public record of an error (e.g., variants of the sparse vector method [11, 12]) or where a seemingly small change to an algorithm breaks an important component of the algorithm (e.g., variants of the noisy max algorithm [24, 30] and the histogram algorithm [31]).

**Programming platforms and verification tools**  Several dynamic tools [32, 33, 34, 35, 36] exist for enforcing differential privacy. Those tools track the privacy budget consumption at runtime, and terminates a program when the intended privacy budget is exhausted. On the other hand, static methods exist for verifying that a program obeys differential privacy during any execution, based on relational program logic [37, 38, 39, 40, 30, 41, 42] and relational type system [43, 44, 45]. We note that those methods are largely orthogonal to this work: their goal is to verify a correct program or to terminate an incorrect one, while our goal is to detect an incorrect program and generate counterexamples for it. The counterexamples provide valuable guidance for fixing incorrect algorithms for algorithm designers. Moreover, we believe our tool fills in the currently missing piece in the development of differentially private algorithms: with our tool, immature designs can first be tested for counterexamples, before being fed into those dynamic and static tools.

Recent work [46, 47, 48] targets both proving and disproving differential privacy. CheckDP [46] relies on the randomness alignment as the underlying proof technique. It reduces the search space of proofs to templates with holes. It embeds a novel bi-directional CEGIS loop to improve proof
and counterexample simultaneously. Barthe et al. [47] identify a non-trivial class of programs where checking (pure and approximate) differential privacy is decidable. However, these programs only allow a bounded number of samples from the Laplace distribution, and their inputs and outputs are from a finite domain. Farina [48] builds a relational symbolic execution framework, which when combined with probabilistic couplings, is able to prove differential privacy for SVT or generate failing traces for its two incorrect variants.

**Counterexample generation** Symbolic execution [49, 50, 51] is widely used for program testing and bug finding. One attractive feature of symbolic execution is that when a property is being violated, it generates counterexamples (i.e., program inputs) that lead to violations. More relevant to this work is work on testing relational properties based on symbolic execution [52, 53, 54]. However, those work only apply to deterministic programs, but the differential privacy property inherently involves probabilistic programs, which is beyond the scope of those work.

In parallel with our work, Bichsel et al. [55] proposed a counterexample generator that uses symbolic differentiation and gradient descent to search for counterexamples. More recently, DP-Sniper [56] trains a classifier – a parametric family of posterior probability distributions to predict if an observed output is likely generated from one of two possible inputs, and use this classifier to select a set of outputs that can best distinguish these two inputs.

**Private selection mechanisms** Selection algorithms, such as Exponential Mechanism [25, 57], Sparse Vector Technique (SVT) [24, 11], and Noisy Max [24] are used to select a set of items (typically queries) from a much larger set. They have applications in hyperparameter tuning [14, 58], iterative construction of microdata [59], feature selection [60], frequent itemset mining [61], exploring a privacy/accuracy tradeoff [62], data pre-processing [63], etc. Various generalizations have been proposed [62, 64, 60, 57, 65, 58]. Liu and Talwar [58] and Raskhodnikova and Smith [57] extend the exponential mechanism for arbitrary sensitivity queries. Beimel et al. [64] and Thakurta and Smith [60] use the propose-test-release framework [66] to find a gap between the best and second best queries and, if the gap is large enough, release the identity of the best query. These two algorithms rely on a relaxation of differential privacy called approximate (ε, δ)-differential privacy [28] and can fail to return an answer (in which case they return ⊥). Our algorithms work with pure ε-differential privacy. Chaudhuri et al. [65] also proposed a
large margin mechanism (with approximate differential privacy) which finds a large gap separating top queries from the rest and returns one of them.

There have also been unsuccessful attempts to generalize selection algorithms such as SVT (incorrect versions are catalogued by Lyu et al. [11]), which has sparked innovations in program verification for differential privacy (e.g., [30, 42, 43, 67]) with techniques such as probabilistic coupling [30] and a simplification based on randomness alignment [43]. These are similar to ideas behind handwritten proofs [14, 24, 11] – they consider what changes need to be made to random variables in order to make two executions of a program, with different inputs, produce the same output. It is a powerful technique that is behind almost all proofs of differential privacy, but is very easy to apply incorrectly [11]. In this dissertation, we state and prove a more general version of this technique in order to prove correctness of our algorithms and also provide additional results that simplify the application of this technique.

**Floating-point vulnerability** The floating point vulnerability in differentially private systems and its severity was first studied by Mironov [23]. As an example, Mironov demonstrated that by examining the low-order bits of the noisy outputs of the Laplace mechanism, the noiseless value can often be determined. As a remedy, Mironov proposed a post-processing snapping procedure which performs rounding and clamping on top of the floating-point arithmetic. However, this leads to inexact sampling probabilities and worse privacy/accuracy guarantees than what is theoretically achievable.

Ghosh et al. [68] proposed a discrete analogue of the Laplace mechanism using the discrete Laplace distribution (a.k.a two-sided geometric distribution). The discrete Laplace mechanism satisfies $\epsilon$-differential privacy and can be implemented on finite computers. It has many desirable properties and is used in the TopDown algorithm for protecting the 2020 US Census data [69, 70], though its implementation does not use an exact sampling procedure [71].

Recently, Cannone et al. proposed a discrete analogue of the Gaussian Mechanism [72] which can be implemented on finite machines with exact sampling. Secure implementations for several other mechanisms are also available, including histogram approximation [73], the exponential mechanism [74] and the Report Noisy Max algorithm [75, 76]. To the best of our knowledge, no secure implementations of our improved selection mechanisms are known.
Chapter 3

Background

3.1 Differential Privacy

We view a database as a finite multiset of records from some domain. It is sometimes convenient to
represent a database by a histogram, where each cell is the count of times a specific record is present.

Differential privacy relies on the notion of adjacent databases. The two most common definitions of
adjacency are: (1) two databases \( D_1 \) and \( D_2 \) are adjacent if \( D_2 \) can be obtained from \( D_1 \) by adding
or removing a single record. (2) two databases \( D_1 \) and \( D_2 \) are adjacent if \( D_2 \) can be obtained from
\( D_1 \) by modifying one record. The notion of adjacency used by an algorithm must be provided to the
counterexample generator. We write \( D_1 \sim D_2 \) to mean that \( D_1 \) is adjacent to \( D_2 \) (under whichever
definition of adjacency is relevant in the context of a given algorithm).

We use the term mechanism to refer to an algorithm \( M \) that tries to protect the privacy of its input.
In our case, a mechanism is an algorithm that is intended to satisfy \( \epsilon \)-differential privacy:

**Definition 3.1 (Differential Privacy [1]).** Let \( \epsilon \geq 0 \). A mechanism \( M \) is said to be \( \epsilon \)-differentially private
if for every pair of adjacent databases \( D_1 \) and \( D_2 \), and every \( E \subseteq \text{Range}(M) \), we have

\[
P(M(D_1) \in E) \leq e^\epsilon \cdot P(M(D_2) \in E).
\]

The value of \( \epsilon \), called the privacy budget, controls the level of the privacy: the smaller \( \epsilon \) is, the more
privacy is guaranteed.

Differential privacy enjoys the following properties:

1. Resilience to Post-Processing. If we apply an algorithm \( A \) to the output of an \( \epsilon \)-differentially private
   algorithm \( M \), then the composite algorithm \( A \circ M \) still satisfies \( \epsilon \)-differential privacy. In other words,
   privacy is not reduced by post-processing.
2. Composition. If \( M_1, M_2, \ldots, M_k \) satisfy differential privacy with privacy loss budgets \( \epsilon_1, \ldots, \epsilon_k \), the algorithm that runs all of them and releases their outputs satisfies \( (\sum_i \epsilon_i) \)-differential privacy.

### 3.2 Global Sensitivity and the Laplace Mechanism

One of the most common building blocks of differentially private algorithms is the Laplace mechanism [1], which is used to answer numerical queries. Let \( D \) be the set of possible databases. A numerical query is a function \( q : D \to \mathbb{R}^k \) (i.e. it outputs a \( k \)-dimensional vector of numbers). The Laplace mechanism is based on a concept called global sensitivity, which measures the worst-case effect one record can have on a numerical query:

**Definition 3.2** (Global Sensitivity [1]). The \( \ell_1 \)-global sensitivity of a numerical query \( q \) is

\[
\Delta_q = \max_{D_1 \sim D_2} \| q(D_1) - q(D_2) \|_1.
\]

The Laplace mechanism works by adding Laplace noise (having density \( f(x; \beta) = \frac{1}{2\beta} e^{-\frac{|x|}{\beta}} \) and variance \( 2\beta^2 \)) to query answers. The chosen variance depends on \( \epsilon \) and the global sensitivity. We use the notation \( \text{Lap}(\beta) \) to refer to the Laplace noise.

**Definition 3.3** (The Laplace mechanism [1]). For any numerical query \( q : D \to \mathbb{R}^n \), the Laplace mechanism outputs

\[
M(D, q, \epsilon) = q(D) + (\eta_1, \ldots, \eta_n)
\]

where \( \eta_i \) are independent random variables sampled from \( \text{Lap}(\Delta_q / \epsilon) \).

**Theorem 3.1** ([24]). The Laplace mechanism is \( \epsilon \)-differentially private.

Other kinds of additive noise distributions that can be used in place of Laplace in Theorem 3.1 include Discrete Laplace [68] (when all query answers are integers or multiples of a common base) and Staircase [77].

In some cases, queries may have additional structure, such as monotonicity, that can allow algorithms to provide privacy with less noise (such as one-sided Noisy Max [24]).
Definition 3.4 (Monotonicity). A list of queries \( q = (q_1, q_2, \ldots) \) with numerical values is monotonic if for all pair of adjacent databases \( D \sim D' \) we have either \( \forall i : q_i(D) \leq q_i(D') \), or \( \forall i : q_i(D) \geq q_i(D') \).

Monotonicity is a natural property that is satisfied by counting queries – when a person is added to a database, the value of each query either stays the same or increases by 1.
Chapter 4

Violation Detection for Privacy Mechanisms

4.1 Introduction

The design of differentially private algorithms is very subtle and error-prone – it is well-known that a large number of published algorithms are incorrect (i.e. they violate differential privacy). A sign of this problem is the existence of papers that are solely designed to point out errors in other papers [11, 12]. The problem is not limited to novices who may not understand the subtleties of differential privacy; it even affects experts whose goal is to design sophisticated algorithms for accurately releasing statistics about data while preserving privacy.

There are two main approaches to tackling this prevalence of bugs: programming platforms and verification. Programming platforms, such as PINQ [32], Airavat [33], and GUPT [78] provide a small set of primitive operations that can be used as building blocks of algorithms for differential privacy. They make it easy to create correct differentially private algorithms at the cost of accuracy (the resulting privacy-preserving query answers and models can become less accurate). Verification techniques, on the other hand, allow programmers to implement a wider variety of algorithms and verify proofs of correctness (written by the developers) [37, 38, 39, 40, 30, 41] or synthesize most (or all) of the proofs [42, 43, 44, 45].

In this chapter, we take a different approach: finding bugs that cause algorithms to violate differential privacy, and generating counterexamples that illustrate these violations. We envision that such a counterexample generator would be useful in the development cycle – variations of an algorithm can be quickly evaluated and buggy versions could be discarded (without wasting the developer’s time in a manual search for counterexamples or a doomed search for a correctness proof). Furthermore, counterexamples can help developers understand why their algorithms fail to satisfy differential privacy and thus can help them fix the problems. This feature is absent in all existing programming platforms
and verification tools. To the best of our knowledge, this is the first paper that treats the problem of detecting counterexamples in incorrect implementations of differential privacy.

Although recent work on relational symbolic execution [79] aims for simpler versions of this task (like detecting incorrect calculations of sensitivity), it is not yet powerful enough to reason about probabilistic computations. Hence, it cannot detect counterexamples in sophisticated algorithms like the sparse vector technique [24], which satisfies differential privacy but is notorious for having many incorrect published variations [11, 12].

Our counterexample generator is designed to function in black-box mode as much as possible. That is, it executes code with a variety of inputs and analyzes the (distribution of) outputs of the code. This allows developers to use their preferred languages and libraries as much as possible; in contrast, most language-based tools will restrict developers to specific programming languages and a very small set of libraries. In some instances, the code may include some tuning parameters. In those cases, we can use an optional symbolic execution model (our current implementation analyzes python code) to find values of those parameters that make it easier to detect counterexamples. Thus, we refer to our method as a semi-black-box approach.

Our contributions are as follows:

• We present the first counterexample generator for differential privacy. It treats programs as semi-black-boxes and uses statistical tests to detect violations of differential privacy.

• We evaluate our counterexample generator on a variety of sophisticated differentially private algorithms and their common incorrect variations. These include the sparse vector method and noisy max [24], which are cited as the most challenging algorithms that have been formally verified so far [42, 30]. In particular, the sparse vector technique is notorious for having many incorrect published variations [11, 12]. We also evaluate the counterexample generator on some simpler algorithms such as the histogram algorithm [80], which are also easy for novices to get wrong (by accidentally using too little noise). In all cases, our counterexample generator produces counterexamples for incorrect versions of the algorithms, thus showing its usefulness to both experts and novices.

• The false positive error (i.e. generating "counterexamples" for correct code) of our algorithm is
controllable because it is based on statistical testing. The false positive rate can be made arbitrarily small just by giving the algorithm more time to run.

Limitations: it is impossible to create counterexample/bug detector that works for all programs. For this reason, our counterexample generator is not intended to be used in an adversarial setting (where a rogue developer wants to add an algorithm that appears to satisfy differential privacy but has a back door). In particular, if a program satisfies differential privacy except with an extremely small probability (a setting known as approximate differential privacy [28]) then our counterexample generator may not detect it. Solving this issue is an area for future work.

The rest of the chapter is organized as follows. Background on statistical testing is discussed in Section 4.2. The counterexample generator is presented in Section 4.3. Experiments are presented in Section 4.4.

4.2 Background

In this section, we discuss relevant background on statistical hypothesis testing.

4.2.1 Hypothesis Testing

A statistical hypothesis is a claim about the parameters of the distribution that generated the data. The null hypothesis, denoted by $H_0$ is a statistical hypothesis that we are trying to disprove. For example, if we have two samples, $X$ and $Y$ where $X$ was generated by a Binomial($n, p_1$) distribution and $Y$ was generated by a Binomial($n, p_2$) distribution, one null hypothesis could be $p_1 = p_2$ (that is, we would like to know if the data supports the conclusion that $X$ and $Y$ came from different distributions). The alternative hypothesis, denoted by $H_1$, is the complement of the null hypothesis (e.g., $p_1 \neq p_2$).

A hypothesis test is a procedure that takes in a data sample $Z$ and either rejects the null hypothesis or fails to reject the null hypothesis. A hypothesis test can have two types of errors: type I and type II. A type I error occurs if the test incorrectly rejects $H_0$ when it is in fact true. A type II error occurs if the test fails to reject $H_0$ when the alternative hypothesis is true. Type I and type II errors are analogous to
false positives and false negatives, respectively.

In most problems, controlling type I error is the most important. In such cases, one specifies a significance level $\alpha$ and requires that the probability of a type I error be at most $\alpha$. Commonly used values for $\alpha$ are 0.05 and 0.01. In order to allow users to control the type I error, the hypothesis test also returns a number $p$ — known as the p-value — which is a probabilistic estimate of how unlikely it is that the null hypothesis is true. The user rejects the null hypothesis if $p \leq \alpha$. In order for this to work (i.e. in order for the Type I error to be below $\alpha$), the p-value must satisfy certain technical conditions:

1. a p-value is a function of a data sample $Z$,
2. $0 \leq p(Z) \leq 1$,
3. if the null hypothesis is true, then $P(p(Z) \leq \alpha | H_0) \leq \alpha$.

A relevant example of a hypothesis test is Fisher’s exact test [81] for two binomial populations. Let $c_1$ be a sample from a Binomial($n_1, p_1$) distribution and let $c_2$ be a sample from a Binomial($n_2, p_2$) distribution. Here $p_1$ and $p_2$ are unknown. Using these values of $c_1$ and $c_2$, the goal is to test the null hypothesis $H_0 : p_1 \leq p_2$ against the alternative $H_1 : p_1 > p_2$. Let $s = c_1 + c_2$. The key insight behind Fisher’s test is that if $C_1 \sim \text{Binomial}(n_1, p_1)$ and $C_2 \sim \text{Binomial}(n_2, p_2)$ and if $p_1 = p_2$, then the value $P(C_1 \geq c_1 | C_1 + C_2 = s)$ does not depend on the unknown parameters $p_1$ or $p_2$ and can be computed from the cumulative distribution function of the hypergeometric distribution; specifically, it is equal to $1 - \text{Hypergeometric.cdf}(c_1 - 1 | n_1 + n_2, n_1, s)$. When $p_1 < p_2$, then $P(C_1 \geq c_1 | C_1 + C_2 = s)$ cannot be computed without knowing $p_1$ and $p_2$. However, it is less than $1 - \text{Hypergeometric.cdf}(c_1 - 1 | n_1 + n_2, n_1, s)$. Thus it can be shown that $1 - \text{Hypergeometric.cdf}(c_1 - 1 | n_1 + n_2, n_1, s)$ is a valid p-value and so the Fisher’s exact test rejects the null hypothesis when this quantity is $\leq \alpha$.

### 4.3 Counterexample Detection

For a mechanism $M$ that does not satisfy $\epsilon$-differential privacy, the goal is to prove this failure. By Definition 3.1, this involves finding a pair of adjacent databases $D_1, D_2$ and an output event $E$ such that $P(M(D_1) \in E) > e^\epsilon P(M(D_2) \in E)$. Thus a counterexample involves finding these two adjacent inputs $D_1$ and $D_2$, the bad output set $E$, and to show that for these choices, $P(M(D_1) \in E) > e^\epsilon P(M(D_2) \in E)$.

Ideally, one would compute the probabilities $P(M(D_1) \in E)$ and $P(M(D_2) \in E)$. Unfortunately, this is read as "$C_1$ is a random variable having the Binomial($n_1, p_1$) distribution".
for sophisticated mechanisms, it is not always possible to compute these quantities exactly. However, we can sample from these distributions many times by repeatedly running $M(D_1)$ and $M(D_2)$ and counting the number of times that the outputs fall into $E$. Then, we need a statistical test to reject the null hypothesis $P(M(D_1) \in E) \leq e^\epsilon P(M(D_2) \in E)$ (or fail to reject it if the algorithm is $\epsilon$-differentially private).

We will be using the following conventions:

- The input to most mechanisms is actually a list of queries $Q = (q_1, \ldots, q_l)$ rather than a database directly. For example, algorithms to release differentially private histograms operate on a histogram of the data; the sparse vector mechanism operates on a sequence of queries that each have global sensitivity equal to 1. Thus, we require the user to specify how the input query answers can differ on two adjacent databases. For example, in a histogram, exactly one cell count changes by at most 1. In the sparse vector technique [24], every query answer changes by at most 1. To simplify the discussion, we abuse notation and use $D_1, D_2$ to also denote the answers of $Q$ on the input adjacent databases. For example, when discussing the sparse vector technique, we write $D_1 = [0, 0]$ and $D_2 = [1, 1]$. This means there are adjacent databases and a list of queries $Q = [q_1, q_2]$ such that they evaluate to $[0, 0]$ on the first database and $[1, 1]$ on the second database.

- We use $\epsilon_0$ to indicate the privacy level that a mechanism claims to achieve.

- We use $\Omega$ for the set of all possible outputs (i.e., range) of the mechanism $M$. We use $\omega$ for a single output of $M$.

- We call a subset $E \subseteq \Omega$ an event. We use $p_1$ (respectively, $p_2$) to denote $P(M(D_i) \in E)$, the probability that the output of $M$ falls into $E$ when executing on database $D_1$ (respectively, $D_2$).

- Some mechanisms take additional inputs, e.g., the sparse vector mechanism. We collectively refer to them as $\text{args}$.

Our discussion is organized as follows. We provide an overview of the counterexample generator in Section 4.3.1. Then we incrementally explain our approach. In Section 4.3.2 we present the hypothesis test. That is, suppose we already have query sequences $D_1$ and $D_2$ that are generated from adjacent databases and an output set $E$, how do we test if $P(M(D_1) \in E) \leq e^\epsilon P(M(D_2) \in E)$ or
\( P(M(D_1) \in E) > e^\epsilon P(M(D_2) \in E) \)? Next, in Section 4.3.3, we consider the question of output selection. That is, suppose we already have query answers \( D_1 \) and \( D_2 \) that are generated from adjacent databases, how do we decide which \( E \) should be used in the hypothesis test? Finally, in Section 4.3.4, we consider the problem of generating the adjacent query sequences \( D_1 \) and \( D_2 \) as well as additional inputs \( args \).

The details of specific mechanisms we test for violations of differential privacy will be given in the experiments in Section 4.4.

### 4.3.1 Overview

At a high level, the counterexample generator can be summarized in the pseudocode in Algorithm 1. First, it generates an InputList, a set of candidate tuples of the form \((D_1, D_2, args)\). That is, instead of returning a single pair of adjacent inputs \( D_1, D_2 \) and any auxiliary arguments the mechanism may need, we return multiple candidates which will be filtered later. Each adjacent pair \((D_1, D_2)\) is designed to be short so that a developer can understand the problematic inputs and trace them through the code of the mechanism \( M \). For this reason, the code of \( M \) will also run fast, so that it will be possible to later evaluate \( M(D_1, args) \) and \( M(D_2, args) \) multiple times very quickly.

**Algorithm 1: Overview of Counterexample Generator**

```plaintext
1 function CounterExampleDetection(M, \epsilon):
   input : M: mechanism
            \epsilon: desired privacy (is M \epsilon-differentially private?)
2   InputList \leftarrow InputGenerator(M, \epsilon)
3   E, D_1, D_2, args \leftarrow EventSelector(M, \epsilon, InputList)
4   p_T, p_\perp \leftarrow HypothesisTest(M, \epsilon, D_1, D_2, args, E)
5   return p_T, p_\perp
```

The next step is the EventSelector. It takes each tuple \((D_1, D_2, args)\) from InputList and runs \( M(D_1, args) \) and \( M(D_2, args) \) multiple times. Based on the type of the outputs, it generates a set of candidates for \( E \). For example, if the output is a real number, then the set of candidates is the set of intervals \((a, b)\). For each candidate \( E \) and each tuple \((D_1, D_2, args)\), it counts how many times \( M(D_1, args) \) produced an output \( \omega \in E \) and how many times \( M(D_2, args) \) produced an output in \( E \).
Based on these results, it picks one specific $E$ and one tuple $(D_1, D_2, \text{args})$ which it believes is most likely to show a violation of $\epsilon_0$-differential privacy.

Finally, the HypothesisTest takes the selected $E, D_1, D_2$, and args and checks if it can detect statistical evidence that $P(M(D_1, \text{args}) \in E) > e^\epsilon P(M(D_2, \text{args}) \in E)$ – which corresponds to the $p$-value $p_\uparrow$ – or $e^\epsilon P(M(D_1, \text{args}) \in E) < P(M(D_2, \text{args}) \in E)$ – which corresponds to the $p$-value $p_\downarrow$.

It is important to note that the EventSelector also uses HypothesisTest internally as a sub-routine to filter out candidates. That is, for every candidate $E$ and every candidate $(D_1, D_2, \text{args})$, it runs HypothesisTest and treats the returned value as a score. The combination of $E$ and $(D_1, D_2, \text{args})$ with the best score is returned by the EvenSelector. Note that EventSelector is using the HypothesisTest in an exploratory way – it evaluates many hypotheses and returns the best one it finds. This is why the $E$ and $(D_1, D_2, \text{args})$ that are finally chosen need to be evaluated again on Line 4 using fresh samples from $M$.

Figure 4-1: Interpreting experimental results on hypothesis tests. A hypothetical algorithm $M$ claims to achieve $\epsilon_0$-differential privacy. For each $\epsilon$ close to $\epsilon_0$ from 0.2 to 2.4 we would evaluate if $M$ satisfies $\epsilon$-differential privacy. We show a typical graph when $M$ possibly provides more privacy (center) and a graph where $M$ provides less privacy than advertised.

**Interpreting the results** One of the best ways of understanding the behavior of the counterexample generator is to look at the $p$-values it outputs. That is, we take an mechanism $M$ that claims to satisfy $\epsilon_0$-differential privacy and, for each $\epsilon$ close to $\epsilon_0$, we test whether it satisfies $\epsilon$-differential privacy (that is, even though $M$ claims to satisfy $\epsilon_0$-differential privacy, we may want to test if it satisfies $\epsilon$-differential privacy for some other value of $\epsilon$ that is close to $\epsilon_0$). The hypothesis tester returns two $p$-values:
• $p_{\top}$. Small values indicate that probably $P(M(D_1, args) \in E) > e^\epsilon P(M(D_2, args) \in E)$.

• $p_{\perp}$. Small values indicate that probably $P(M(D_2, args) \in E) > e^\epsilon P(M(D_1, args) \in E)$.

For each $\epsilon$, we plot the minimum of $p_{\top}$ and $p_{\perp}$. Figure 4-1 shows typical results that would appear when the counterexample detector is run with real mechanisms $M$ as input.

In Figure 4-1a, $M$ correctly satisfies the claimed $\epsilon_0 = 0.7$ differential privacy. In that plot, we see that the $p$-values corresponding to $\epsilon = 0.2, 0.4, 0.6$ are very low, meaning that the counterexample generator can prove that the algorithm does not satisfy differential privacy for those smaller values of $\epsilon$. Near 0.7 it becomes difficult to find counterexamples; that is, if an algorithm satisfies 0.7-differential privacy, it is very hard to statistically prove that it does not satisfy 0.65 differential privacy. This is a typical feature of hypothesis tests as it becomes difficult to reject the null hypothesis when it is only slightly incorrect (e.g., when the true privacy parameter is only slightly different from the $\epsilon$ we are testing). Now, any algorithm that satisfies 0.7-differential privacy also satisfies $\epsilon$-differential privacy for all $\epsilon \geq 0.7$. This behavior is seen in Figure 4-1a as the $p$-values are large for all larger values of $\epsilon$.

Figure 4-1b shows a graph that can arise from two distinct scenarios. One of the situations is when the mechanism $M$ claims to provide 0.7-differential privacy but actually provides more privacy (i.e. $\epsilon$-differential privacy for $\epsilon < 0.7$). In this figure, the counterexample generator could prove, for example, that $M$ does not satisfy 0.4-differential privacy, but leaves open the possibility that it satisfies 0.5-differential privacy. The other situation is when our tool has failed to find good counterexamples. Thus when a mechanism is correct, good precision by the counterexample generator means that the line starts rising close to (but before the dotted line), and worse precision means that the line starts rising much earlier.

Figure 4-1c shows a typical situation in which an algorithm claims to satisfy 0.7-differential privacy but actually provides less privacy than advertised. In this case, the counterexample generator can generate good counterexamples at $\epsilon = 0.7$ (the dotted line) and even at much higher values of $\epsilon$. When an mechanism is incorrect, such a graph indicates good precision by the counterexample generator.

**Limitations** In some cases, finding counterexamples requires a large input datasets. In those cases, searching for the right inputs and running algorithms on them many times will impact the ability of
our counterexample generator to find counterexamples. This is a limitation of all techniques based on statistical tests.

Another important case where our counterexample generator is not expected to perform well is when violations of differential privacy happen very rarely. For example, consider a mechanism $M$ that checks if its input is $D_1 = [1]$. If so, with probability $e^{-9}$ it outputs 1 and otherwise it outputs 0 (if the input is not 1, $M$ always outputs 0). $M$ does not satisfy $\varepsilon$-differential privacy for any value of $\varepsilon$. However, showing it statistically is very difficult. Supposing $D_1 = [1]$ and $D_2 = [0]$ are adjacent databases, it requires running $M(D_1)$ and $M(D_2)$ billions of times to observe that an output of 1 is possible under $D_1$ but is at least $e^{\varepsilon}$ times less likely under $D_2$.

Addressing both of these problems will likely involve incorporation of program analysis, such as symbolic execution, into our statistical framework and is a direction for future work.

### 4.3.2 Hypothesis Testing

#### Algorithm 2: Hypothesis Test. Parameter $n$: # of iterations

```plaintext
1 function pvalue($c_1$, $c_2$, $n$, $\varepsilon$):
    $\tilde{c}_1 \leftarrow \text{Binomial}(c_1, 1/e^\varepsilon)$
    $s \leftarrow \tilde{c}_1 + c_2$
    return $1 - \text{Hypergeom.cdf}(\tilde{c}_1 - 1 | 2n, n, s)$

5 function HypothesisTest($n$, $M$, $args$, $\varepsilon$, $D_1$, $D_2$, $E$):
    input: $M$: mechanism
    $args$: additional arguments for $M$
    $\varepsilon$: privacy budget to test
    $D_1$, $D_2$: adjacent databases
    $E$: Event
    $O_1 \leftarrow$ results of running $M(D_1, args)$ for $n$ times
    $O_2 \leftarrow$ results of running $M(D_2, args)$ for $n$ times
    $c_1 \leftarrow |\{i | O_1[i] \in E\}|$
    $c_2 \leftarrow |\{i | O_2[i] \in E\}|$
    $p_\top \leftarrow pvalue(c_1, c_2, n, \varepsilon)$
    $p_\bot \leftarrow pvalue(c_2, c_1, n, \varepsilon)$
    return $p_\top, p_\bot$
```

Suppose we have a mechanism $M$, inputs $D_1, D_2$ and an output set $E$ (we discuss the generation of $D_1, D_2$ in Section 4.3.4 and $E$ in Section 4.3.3). We would like to check if $P(M(D_1) \in E) >$
where $X$ (which implies that in the border case, the other case is symmetric.

The challenge is, of course, in the last step as we don’t know what distribution. The marginal distribution of $Z$ is

Let $\tilde{c}_1 \sim \text{Binomial}(c_1, 1/e^\epsilon)$ sample $\tilde{p}_1$ and $\tilde{p}_2$ (the higher the variance, the less the test should trust the estimates).

Instead, we take a different approach that allows us to conduct the test without knowing what $p_1$ and $p_2$ are. First, we note that $c_1$ and $c_2$ are equivalent to samples from a Binomial($n, p_1$) distribution and a Binomial($n, p_2$) distribution respectively. We first consider the border case where $p_1 = e^\epsilon p_2$. Consider sample $\tilde{c}_1$ from a Binomial($c_1, 1/e^\epsilon$) distribution. We note that this sample enjoys the following property (which implies that in the border case, $\tilde{c}_1$ has the same distribution as $c_2$):

**Lemma 4.1.** Let $X \sim \text{Binomial}(n, p_1)$ and $Z$ be generated from $X$ by sampling from the Binomial($X, \frac{1}{e^\epsilon}$) distribution. The marginal distribution of $Z$ is Binomial($n, \frac{p_1}{e^\epsilon}$).

**Proof.** The relationship between Binomial and Bernoulli random variables means that $X = \sum_{i=1}^{n} X_i$, where $X_i$ is a Bernoulli($p_1$) random variable. Generating $Z$ from $X$ is the same as doing the following: set $Z_i = 0$ if $X_i = 0$. If $X_i = 1$, set $Z_i = 1$ with probability $1/e^\epsilon$ (and set $Z_i = 0$ otherwise). Then set $Z = \sum_{i=1}^{n} Z_i$. Hence, the marginal distribution of $Z_i$ is a Bernoulli($p_1/e^\epsilon$) random variable:

\[
P(Z_i = 1) = P(Z_i = 1 | X_i = 1)P(X_i = 1) + P(Z_i = 1 | X_i = 0)P(X_i = 0)
= (1/e^\epsilon) \cdot p_1 + 0 \cdot (1 - p_1) = p_1/e^\epsilon
\]
This means that the marginal distribution of $Z$ is Binomial$(n, p_1/e^\epsilon).$ \hfill \qed

Thus we have the following facts that follow immediately from the lemma:

- If $p_1 > e^\epsilon p_2$ then the distribution of $\tilde{c}_1$ is Binomial$(n, \tilde{p}_1)$ with $\tilde{p}_1 = p_1/e^\epsilon$ and so has a larger Binomial parameter than $c_2$ (which is Binomial$(n, p_2)$). We want our test to be able to reject the null hypothesis in this case.

- If $p_1 = e^\epsilon p_2$ then the distribution of $\tilde{c}_1$ is Binomial$(n, \tilde{p}_1)$ with $\tilde{p}_1 = p_2$ and so has the same Binomial parameter as $c_2$. We do not want our test to reject the null hypothesis in this case.

- If $p_1 < e^\epsilon p_2$ then the distribution of $\tilde{c}_1$ is Binomial$(n, \tilde{p}_1)$ with $\tilde{p}_1 = p_1/e^\epsilon$ and so has a smaller Binomial parameter than $c_2$ (which is Binomial$(n, p_2)$). We do not want to reject the null hypothesis in this case.

Thus, by randomly generating $\tilde{c}_1$ from $c_1$, we have (randomly) reduced the problem of testing $p_1 > e^\epsilon p_2$ vs. $p_1 \leq e^\epsilon p_2$ (on the basis of $c_1$ and $c_2$) to the problem of testing $\tilde{p}_1 > p_2$ vs. $\tilde{p}_1 \leq p_2$ (on the basis of $\tilde{c}_1$ and $c_2$). Now, checking whether $\tilde{c}_1$ and $c_2$ come from the same distribution can be done with the Fisher’s exact test (see Section 4.2): the $p$-value is $1 - \text{Hypergeom.cdf}(\tilde{c}_1 - 1 \mid 2n, n, \tilde{c}_1 + c_2).$\footnote{Here we use a notation from SciPy [82] package where Hypergeom.cdf means the cumulative distribution function of hypergeometric distribution.}

This is done in the function $pvalue$ in Algorithm 2.

To summarize, given $c_1$ and $c_2$, we first sample $\tilde{c}_1$ from the Binomial$(c_1, 1/e^\epsilon)$ distribution and then return the $p$-value of $(1 - \text{Hypergeom.cdf}(\tilde{c}_1 - 1 \mid 2n, n, \tilde{c}_1 + c_2))$. Since this is a random reduction, we reduce its variance by sampling $\tilde{c}_1$ multiple times and averaging the $p$-values. That is, we run the $p$-value function (Algorithm 2) multiple times with the same inputs and average the $p$-values it returns.

### 4.3.3 Event Selection

Having discussed how to test if $P(M(D_1) \in E) > e^\epsilon P(M(D_2) \in E)$ or if $P(M(D_2) \in E) > e^\epsilon P(M(D_1) \in E)$ when $D_1, D_2,$ and $E$ were pre-specified, we now discuss how to select the event $E$ that is most likely to show violations of $\epsilon$-differential privacy.
Algorithm 3: Event Selector. Parameter $n$: # of iterations

1. function EventSelector($n$, $M$, $\epsilon$, InputList):
   2. input : $M$: mechanism
      3. $\text{InputList}$: possible inputs
      4. $\epsilon$: privacy budget to test
      5. $pvalues \leftarrow []$
      6. $results \leftarrow []$
      7. foreach ($D_1$, $D_2$, $args$) $\in$ InputList do
         8. $\text{SearchSpace} \leftarrow$ search space based on return type
         9. $O_1 \leftarrow$ results of running $M(D_1, args)$ for $n$ times
        10. $O_2 \leftarrow$ results of running $M(D_2, args)$ for $n$ times
        11. foreach $E \in \text{SearchSpace}$ do
            12. $c_1 \leftarrow |\{i \mid O_1[i] \in E\}|$
            13. $c_2 \leftarrow |\{i \mid O_2[i] \in E\}|$
            14. $p_\top \leftarrow \text{pvalue}(c_1, c_2, n, \epsilon)$
            15. $p_\bot \leftarrow \text{pvalue}(c_2, c_1, n, \epsilon)$
            16. $pvalues$.append($\min(p_\top, p_\bot)$)
            17. $results$.append($D_1$, $D_2$, $args$, $E$)
        18. return $results[\arg\min(pvalues)]$

One of the challenges is that different mechanisms could have different output types (e.g., a discrete number, a vector of numbers, a vector of categorical values, etc.). To address this problem, we define a search space $S$ of possible events to look at. The search space depends on the type of the output $\omega$ of $M$, which can be determined by running $M(D_1)$ and $M(D_2)$ multiple times.

1. **The output $\omega$ is a fixed length list of categorical values.** We first run $M(D_1)$ once and ask it to not use any noise (i.e. tell it to satisfy $\epsilon$-differential privacy with $\epsilon = \infty$). Denote this output as $\omega_0$. Now, when $M$ runs with its preferred privacy settings to produce an output $\omega$, we define $t(\omega)$ be the Hamming distance between the output $\omega$ and $\omega_0$. The search space is

   \[
   S = \{\omega \mid t(\omega) = k : k = 0, 1, \ldots, l\}
   \]

   where $l$ is the fixed length of output of $M$. Another set of events relate to the count of a categorical value in the output. If there are $m$ values, then define

   \[
   S_i = \{\omega \mid \omega.\text{count}(value_i) = k : k = 0, 1, \ldots, l\},
   \]
1 ≤ i ≤ m. The overall search space is the union of S and all S_i.

2. The output ω is a variable length list of categorical values.  In this case, one extra set of events E we look at correspond to the length of the output. For example, we may check if \( P(M(D_1) \text{ has length } k) > P(M(D_2) \text{ has length } k) \). Hence, we define

\[
S_0 = \{ \{ \omega \mid \omega.\text{length} = k \} : k = 0, 1, \ldots \}
\]

For the search space S, we use this \( S_0 \) unioned with the search space from the previous case.

3. The output ω is a fixed length list of numeric values.

In this case, the output is of the form \( \omega = (a_1, \ldots, a_m) \). Our search space is the union of the following:

\[
\{ \{ \omega \mid \omega[i] \in (a, b) \} : i = 1, \ldots, m \text{ and } a < b \},
\]

\[
\{ \{ \omega \mid \text{avg}(\omega) \in (a, b) \} : a < b \},
\]

\[
\{ \{ \omega \mid \text{min}(\omega) \in (a, b) \} : a < b \},
\]

\[
\{ \{ \omega \mid \text{max}(\omega) \in (a, b) \} : a < b \}.
\]

That is, we would end up checking if \( P(\text{avg}(M(D_1)) \in (a, b)) > P(\text{avg}(M(D_2)) \in (a, b)) \), etc.

To save time, we often restrict \( a \) and \( b \) to be multiples of a small number like ±0.2, or ±∞. In the case that the output ω is always an integer array, we replace the condition “∈ (a, b)” with “= k” for each integer \( k \).

4. \( M \) outputs a variable length list of numeric values.

The search space is the union of Case 3 and \( S_0 \) in Case 2.

5. \( M \) outputs a variable length list of mixed categorical and numeric values.  In this case, we separate out the categorical values from numeric values and use the cross product of the search spaces for numeric and categorical values. For instance, events would be of the form “\( \omega \) has \( k \) categorical components equal to \( \ell \) and the average of the numerical components of \( \omega \) is in \( (a, b) \)”
The EventSelector is designed to return one event $E$ for use in the hypothesis test in Algorithm 1. The way EventSelector works is it receives an InputList, which is a set of tuples $(D_1, D_2, \text{args})$ where $D_1, D_2$ are adjacent databases and args is a set of values for any other parameters $M$ needs. For each such tuple, it runs $M(D_1)$ and $M(D_2)$ for $n$ times each. Then for each possible event in the search space, it runs the hypothesis test (as an exploratory tool) to get a p-value. The combination of $(D_1, D_2, \text{args})$ and $E$ that produces the lowest p-value is then returned to Algorithm 1. Algorithm 1 uses those choices to run the real hypothesis test on fresh executions of $M$ on $D_1$ and $D_2$.

The pseudocode for the EventSelector is shown in Algorithm 3.

### 4.3.4 Input Generation

**Algorithm 4**: Input Generator.

1. function $\text{ArgumentGenerator}(M, D_1, D_2)$:
   2. $\text{args}_0 \leftarrow$ Arguments used in noise generation with values that minimize the noises
   3. $\text{constraints} \leftarrow$ Traverse the source code of $M$ and generate constraints to force $D_1$ and $D_2$ to diverge on branches
   4. $\text{args}_1 \leftarrow \text{MaxSMT}(\text{constraints})$
   5. return $\text{args}_0 + \text{args}_1$

6. function $\text{InputGenerator}(M, \text{len})$:
   7. input : $M$: mechanism
   8. len: length of input to generate
   9. $\text{candidates} \leftarrow$ Empirical pairs of databases of length len
   10. $\text{InputList} \leftarrow []$
   11. foreach $(D_1, D_2) \in \text{candidates}$ do
        12. $\text{args} \leftarrow \text{ArgumentGenerator}(M, D_1, D_2)$
        13. $\text{InputList}.\text{append}(D_1, D_2, \text{args})$
   14. return $\text{InputList}$

In this section we discuss our approaches for generating candidate tuples $(D_1, D_2, \text{args})$ where $D_1, D_2$ are adjacent databases and args is a set of auxiliary parameters that a mechanism $M$ may need.

---

3In practice, to avoid choosing bad $E$, we let $c_E$ be the total number of times $M(D_1)$ and/or $M(D_2)$ produced an output in $E$. Then it only executes Line 11-14 in Algorithm 3 if $c_E \geq 0.001 \cdot n \cdot e^\epsilon$, otherwise the selection of $E$ is too noisy.
Database Generation

To find the adjacent databases that are likely to form the basis of counterexamples that illustrate violations of differential privacy, we adopt a simple and generic approach that works surprisingly well. Recalling that the inputs to mechanisms are best modeled as a vector of query answers, we use the type of patterns shown in Table 4-1.

Table 4-1: Database Categories and Samples

<table>
<thead>
<tr>
<th>Category</th>
<th>Sample D1</th>
<th>Sample D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Above</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[2, 1, 1, 1, 1]</td>
</tr>
<tr>
<td>One Below</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[0, 1, 1, 1, 1]</td>
</tr>
<tr>
<td>One Above Rest Below</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[2, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>One Below Rest Above</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[0, 2, 2, 2, 2]</td>
</tr>
<tr>
<td>Half Half</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[0, 0, 0, 2, 2]</td>
</tr>
<tr>
<td>All Above &amp; All Below</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[2, 2, 2, 2, 2]</td>
</tr>
<tr>
<td>X Shape</td>
<td>[1, 1, 0, 0, 0]</td>
<td>[0, 0, 1, 1, 1]</td>
</tr>
</tbody>
</table>

The “One Above” and “One Below” categories are suitable for algorithms whose input is a histogram (i.e. in adjacent databases, at most one query can change, and it will change by at most 1). The rest of the categories are suitable when in adjacent databases every query can change by at most one (i.e. the queries have sensitivity $\Delta_q = 1$).

The design of the categories is based on the wide variety of changes in query answers that are possible when evaluated on one database and on an adjacent database. For example, it could be the case that a few of the queries increase (by 1, if their sensitivity is 1, or by $\Delta_q$ in the general case) but most of them decrease. A simple representative of this situation is “One Above Rest Below” in which one query increases and the rest decrease. The category “One Below Rest Above” is the reverse.

Another situation is where roughly half of the queries increase and half decrease (when evaluated on a database compared to when evaluated on an adjacent database). This scenario is captured by the “Half Half” category. Another situation is where all of the queries increase. This is captures by the “All Above & All Below” category. Finally, the “X Shape” category captures the setting where the query answers are not all the same and some increase and others decrease when evaluated on one database compared to

---

$^4$For queries with larger sensitivity, the extension is obvious. For example $D_1 = [1, 1, 1, 1, 1]$ and $D_2 = [1 + \Delta_q, 1 + \Delta_q, ..., 1 + \Delta_q]$
an adjacent database.

These categories were chosen from our desire to allow counterexamples to be easily understood by mechanism designers (and to make it easier for them to manually trace the code to understand the problems). Thus the samples are short and simple. We consider inputs of length 5 (as in Table 4-1) and also versions of length 10.

**Argument Generation**

Some differentially-private algorithms require extra parameters beyond the database. For example, the sparse vector technique [24], shown in Algorithm 11, takes as inputs a threshold $T$ and a bound $N$. It tries to output numerical queries that are larger than $T$. However, for privacy reasons, it will stop after it returns $N$ noisy queries whose values are greater than $T$. These two arguments are specific to the algorithm and their proper values depend on the desired privacy level as well as algorithm precision.

To find values of auxiliary parameters (such as $N$ and $T$ in Sparse Vector), we build argument generator based on *Symbolic Execution* [83], which is typically used for bug finding: it generates concrete inputs that violate assertions in a program. In general, a symbolic executor assigns symbolic values, rather than concrete values as normal execution would do, for inputs. As the execution goes, the executor maintains a symbolic program state at each assertion and generates constraints that will violate the assertion. When those constraints are satisfiable, concrete inputs (i.e., a solution of the constraints) are generated.

Compared with standard symbolic execution, a major difference in our argument generation is that we are interested in algorithm arguments that will likely maximize the privacy cost of an algorithm. In other words, there is no obvious assertion to be checked in our argument generation. To proceed, we use two heuristics that likely will cause large privacy cost of an algorithm:

- The first heuristic applies to parameters that affect noise generation. For example in Sparse Vector, the algorithm adds $\text{Lap}(2 \cdot N \cdot \Delta_q / \epsilon_0)$ noise. For such a variable, we use the value that results in small amount of noise (i.e., $N = 1$). Small amount of noise is favorable since it reduces the variance in the hypothesis testing (Section 4.3.2).

- The second heuristic (for variables that do not affect noise) prefers arguments that make two program
executions using two different databases (as described in Section 12) to take as many diverging branches as possible. The reason is that diverging branches will likely use more privacy budget.

Next, we give a more detailed overview of our customized symbolic executor. The symbolic executor takes a pair of *concrete* databases as inputs (as described in Section 12) and uses *symbolic* values for other input parameters. Random samples in the program (e.g., a sample from Laplace distribution) are set to value 0 in the symbolic execution. Then, the symbolic executor tracks symbolic program states along program execution in the standard way [83]. For example, the executor will generate a constraint\(^5\) \(x = y + 1\) after an assignment \((x \leftarrow y + 1)\), assuming that variable \(y\) has a symbolic value \(y\) before the assignment. Also, the executor will unroll loops in the source code, which is standard in most symbolic executors.

Unlike standard symbolic executors, the executor conceptually tracks a pair of symbolic program states along program execution (one on concrete database \(D_1\), and one on concrete database \(D_2\)). Moreover, it also generate extra constraints, according to the two heuristics above, in the hope of maximizing the privacy cost of an algorithm. In particular, it handles two kinds of statements in the following way:

- **Sampling.** The executor generates two constraints for a sampling statement: a constraint that eliminates randomness in symbolic execution by assigning sample to value 0, and a constraint that ensures a small amount of noise. Consider a statement \((\eta \leftarrow Lap(e))\). The executor generates two constraints: \(\eta = 0\) as well as a constraint that minimizes expression \(e\).

- **Branch.** The executor generates a constraint that makes the two executions diverge on branches. Consider a branch statement \((if\ e\ then\ \cdots)\). Assume that the executor has symbolic values \(e_1\) and \(e_2\) for the value of expression \(e\) on databases \(D_1\) and \(D_2\) respectively; it will generates a constraint \((e_1 \land \neg e_2) \lor (\neg e_1 \land e_2)\) to make the executions diverge. Note that unlike other constraints, a diverging constraint might be unsatisfiable (e.g., if the query answers under \(D_1\) and \(D_2\) are the same). However, our goal is to *maximize* the number of satisfiable diverging constraints, which can be achieved by a MaxSMT solver.

\(^5\)For simplicity, we use a simple representation for constraints; Z3 has an internal format and a user can either use Z3’s APIs or SMT2 [84] format to represent constraints.
The executor then uses an external MaxSMT solver such as Z3 [85] on all generated constraints to find arguments that maximizes the number of diverged branches.

For example, the correct version of the Sparse Vector algorithm (see the complete algorithm in Algorithm 11) has the parameter $T$ (a threshold). It has a branch that tests whether the noisy query answer is above the threshold $T$:

\[ q + \eta_2 \geq \hat{T} \]

Here, $\eta_2$ is a noise variable, $q$ is one query answer (i.e. one of the components of the input $D_1$ of the algorithm) and $\hat{T}$ is a noisy threshold ($\hat{T} = T + \eta_1$). Suppose we start from a database candidate ([1, 1, 1, 1, 1], [2, 2, 2, 2, 2]). The symbolic executor assigns symbolic values to the parameters $T$ and unrolls the loop in the algorithm, where each iteration handles one noisy query. Along the execution, it updates program states. For example, statement $\hat{T} \leftarrow T + \eta_1$ results in $\hat{T} = T + \eta_1$. For the first execution of the branch of interest, the executor tracks the following symbolic program state:

\[ q_1 = 1 \land q_2 = 2 \land \eta_1 = 0 \land \eta_2 = 0 \land \hat{T}_1 = T + \eta_1 \land \hat{T}_2 = T + \eta_2 \]

as well as the following constraint for diverging branches:

\[ (q_1 + \eta_1 \geq \hat{T}_1 \land q_2 + \eta_2 < \hat{T}_2) \lor (q_1 + \eta_1 < \hat{T}_1 \land q_2 + \eta_2 \geq \hat{T}_2) \]

Similarly, the executor generates constraints from other iterations. In this example, the MaxSMT solver returns a value in between of 1 and 2 so that constraints from all iterations are satisfied. This value of $T$ is used as $\text{arg}$ in the candidate tuple $(D_1, D_2, \text{arg})$.

4.4 Experiments

We implemented our counterexample detection framework with all components, including hypothesis test, event selector and input generator. The implementation is publicly available\(^6\). The tool takes in an algorithm implementation and the desired privacy bound $\epsilon_0$, and generates counterexamples if the

Figure 4-2: Results of Noisy Max algorithm and its variants.

algorithm does not satisfy $\epsilon_0$-differential privacy.

In this section we evaluate our detection framework on some of the popular privacy mechanisms and their variations. We demonstrate the power of our tool: for mechanisms that falsely claim to be differentially private, our tool produces convincing evidence that this is not the case in just a few seconds.

4.4.1 Noisy Max

Report Noisy Max reports which one among a list of counting queries has the largest value. It adds $\text{Lap}(2/\epsilon_0)$ noise to each answer and returns the index of the query with the largest noisy answer. The correct versions have been proven to satisfy $\epsilon_0$-differential privacy [24] no matter how long the input list
is. A naive proof would show that it satisfies \((\epsilon_0 \cdot |Q|/2)\)-differential privacy (where \(|Q|\) is the length of the input query list), but a clever proof shows that it actually satisfies \(\epsilon_0\)-differential privacy.

**Algorithm 5: Correct Noisy Max with Laplace noise**

```plaintext
1 function NoisyMax(Q, \epsilon_0):
   input : Q: queries to the database, \(\epsilon_0\): privacy budget.
2 NoisyVector ← []
3 for \(i = 1 \ldots \text{len}(Q)\) do
4   NoisyVector[i] ← \(Q[i] + \text{Lap}(2/\epsilon_0)\)
5 return \(\text{argmax}(\text{NoisyVector})\)
```

**Adding Laplace Noise**

The correct Noisy Max algorithm (Algorithm 5) adds independent \(\text{Lap}(2/\epsilon_0)\) noise to each query answer and returns the index of the maximum value. As Figure 4-2a shows, we test this algorithm for different privacy budget \(\epsilon_0\) at 0.2, 0.7, 1.5. All lines rise when the test \(\epsilon\) is slightly less than the claimed privacy level \(\epsilon_0\) of the algorithm. This demonstrates the precision of our tool: before \(\epsilon_0\), there is almost 0 chance to falsely claim that this algorithm is not private; after \(\epsilon_0\), the \(p\)-value is too large to conclude that the algorithm is incorrect. We note that the test result is very close to the ideal cases, illustrated by the vertical dashed lines.

**Algorithm 6: Correct Noisy Max with Exponential noise**

```plaintext
1 function NoisyMax(Q, \epsilon_0):
   input : Q: queries to the database, \(\epsilon_0\): privacy budget.
2 NoisyVector ← []
3 for \(i = 1 \ldots \text{len}(Q)\) do
4   NoisyVector[i] ← \(Q[i] + \text{Exponential}(2/\epsilon_0)\)
5 return \(\text{argmax}(\text{NoisyVector})\)
```

**Adding Exponential Noise**

One correct variant of Noisy Max adds \(\text{Exponential}(2/\epsilon_0)\) noise, rather than Laplace noise, to each query answer (Algorithm 6). This mechanism has also been proven to be \(\epsilon_0\)-differential private[24]. Figure 4-2b shows the corresponding test result, which is similar to that of Figure 4-2a. The result indicates that this correct variant likely satisfies \(\epsilon_0\)-differential privacy for the claimed privacy budget.
Algorithm 7: Incorrect Noisy Max with Laplace noise, returning the maximum value

```
1 function NoisyMax(Q, \(\epsilon_0\)):
    2     input : Q: queries to the database, \(\epsilon_0\): privacy budget.
    3     NoisyVector ← []
    4     for i = 1 \ldots len(Q) do
    5             NoisyVector[i] ← Q[i] + Laplace(2/\(\epsilon_0\))
    6     // returns maximum value instead of index
    7     return \text{max}(\text{NoisyVector})
```

Algorithm 8: Incorrect Noisy Max with Exponential noise, returning the maximum value

```
1 function NoisyMax(Q, \(\epsilon_0\)):
    2     input : Q: queries to the database, \(\epsilon_0\): privacy budget.
    3     NoisyVector ← []
    4     for i = 1 \ldots len(Q) do
    5             NoisyVector[i] ← Q[i] + Exponential(2/\(\epsilon_0\))
    6     // returns maximum value instead of index
    7     return \text{max}(\text{NoisyVector})
```

Incorrect Variants of Exponential Noise

An incorrect variant of \textit{NoisyMax} has the same setup but instead of returning the \textit{index} of maximum value, it directly returns the maximum value. We evaluate on two variants that report the maximum value instead of the index (Algorithm 7 and 8) and show the test result in Figure 4-2c and 4-2d.

For the variant using Laplace noise (Figure 4-2c), we can see that for \(\epsilon_0 = 0.2\), the line rises at around test \(\epsilon\) of 0.4, indicating that this algorithm is incorrect for the claimed privacy budget of 0.2. The same pattern happens when we set privacy budget to be 0.7 and 1.5: all lines rise much later than their claimed privacy budget. In this incorrect version, returning the maximum value (instead of its index) causes the algorithm to actually satisfy \(\epsilon_0 \cdot |Q|/2\) differential privacy instead of \(\epsilon_0\)-differential privacy.

For the variant using Exponential noise (Figure 4-2d), the lines rise much later than the claimed privacy budgets, indicating strong evidence that this variant is indeed incorrect. Also, we can hardly see the lines for privacy budgets 0.7 and 1.5, since their p-values remain 0 for all the test \(\epsilon\) ranging from 0 to 2.2 in the experiment.
Algorithm 9: Histogram

1 function Histogram(Q, $\epsilon_0$):
   input : Q: queries to the database, $\epsilon_0$: privacy budget.
2 NoisyVector ← [ ]
3 for $i = 1 \ldots \text{len}(Q)$ do
4   NoisyVector[$i$] ← $Q[i] + \text{Lap}(1/\epsilon_0)$
5 return NoisyVector

Algorithm 10: Histogram with wrong scale

1 function Histogram(Q, $\epsilon_0$):
   input : Q: queries to the database, $\epsilon_0$: privacy budget.
2 NoisyVector ← [ ]
3 for $i = 1 \ldots \text{len}(Q)$ do
4   // wrong scale of noise is added
5   NoisyVector[$i$] ← $Q[i] + \text{Lap}(\epsilon_0)$
6 return NoisyVector

4.4.2 Histogram

The Histogram algorithm [80] is a very simple algorithm for publishing an approximate histogram of
the data. The input is a histogram and the output is a noisy histogram with the same dimensions. The
Histogram algorithm requires input queries to differ in at most one element. Here we evaluate with
different scale parameters for the added Laplace noise.

The correct Histogram algorithm adds independent $\text{Lap}(1/\epsilon_0)$ noise to each query answer, as shown
in Algorithm 9. Since at most one query answer may differ by at most 1, returning the maximum value
is $\epsilon_0$-differentially private [80].

To mimic common mistakes made by novices of differential privacy, we also evaluate on an incorrect
variant where $\text{Lap}(\epsilon_0)$ noise is used in the algorithm (Algorithm 10). We note that the incorrect variant
here satisfies $1/\epsilon_0$-differential privacy, rather the claimed $\epsilon_0$-differential privacy.

Figures 4-3a and 4-3b show the test results for the correct and incorrect variants respectively. Here,
Figures 4-3a indicates that the correct implementation satisfies the claimed privacy budgets. For the
incorrect variant, the claimed budgets of 0.2 and 0.7 are correctly rejected; this is expected since the
true privacy budgets are $1/0.2$ and $1/0.7$ respectively for this incorrect version. Interestingly, the result
indicates that for $\epsilon_0 = 1.5$, this algorithm is likely to be more private than claimed (the line rise around 0.6
(a) Correct Histogram algorithm with \( \text{Lap}(1/\epsilon_0) \) noise.

(b) Incorrect Histogram algorithm with \( \text{Lap}(\epsilon_0) \) noise. It provides more privacy than advertised when \( \epsilon_0 \geq 1 \) and less privacy than advertised when \( \epsilon_0 < 1 \).

Figure 4-3: Results of Histogram algorithm and its variants

rather than 1.5). Again, this is expected since in this case, the variant is indeed \( 1/1.5 = 0.67 \)-differentially private.

### 4.4.3 Sparse Vector

The Sparse Vector Technique (SVT) [86] (see Algorithm 11) is a powerful mechanism for answering numerical queries. It takes a list of numerical queries and simply reports whether their answers are above or below a preset threshold \( T \). It allows the program to output some noisy query answers without any privacy cost. In particular, arbitrarily many “below threshold” answers can be returned, but only at most \( N \) “above threshold” answers can be returned. Because of this remarkable property, there are many variants proposed in both published papers and practical use. However, most of them turn out to be actually not differentially private[11]. We test our tool on a correct implementation of SVT and the major incorrect variants summarized in [11]. In the following, we describe what the variants do and list their pseudocodes.

```
input : \( Q \): queries to the database, \( \epsilon_0 \): privacy budget
\( T \): threshold, \( N \): bound of outputting True’s
\( \Delta \): sensitivity

function SVT(\( Q, T, \epsilon_0, \Delta, N \)):
    \( \text{out} \leftarrow [] \)
    \( \eta_1 \leftarrow \text{Lap}(2 \ast \Delta/\epsilon_0) \)
    \( \tilde{T} \leftarrow T + \eta_1 \)
    \( \text{count} \leftarrow 0 \)
    foreach \( q \) in \( Q \) do
        \( \eta_2 \leftarrow \text{Lap}(4 \ast N \ast \Delta/\epsilon_0) \)
        if \( q + \eta_2 \geq \tilde{T} \) then
            \( \text{out} \leftarrow \text{True} :: \text{out} \)
            \( \text{count} \leftarrow \text{count} + 1 \)
            if \( \text{count} \geq N \) then
                Break
            else
                \( \text{out} \leftarrow \text{False} :: \text{out} \)
        return (\( \text{out} \))
```

SVT [11]

Lyu et al. have proposed an implementation of SVT and proved that it satisfies \( \epsilon_0 \)-differential privacy. This algorithm (Algorithm 11) tries to allocate the global privacy budget \( \epsilon_0 \) into two parts: half of the privacy budget goes to the threshold, and the other half goes to values which are above the threshold. There will not be any privacy cost if the noisy value is below the noisy threshold, in which case the program will output a False. If the noisy value is above the noisy threshold, the program will output a True. After outputting a certain amount (\( N \)) of True’s, the program will halt.

Figure 4-4a shows the test result for this correct implementation. All lines rise around the true privacy budget, indicating that our tool correctly conclude that this algorithm is correct.

iSVT 1 [87]

One incorrect variant (Algorithm 12) adds no noise to the query answers, and has no bound on the number of True’s that the algorithm can output. This implementation does not satisfy \( \epsilon_0 \)-differential privacy for any finite \( \epsilon_0 \).
Algorithm 12: iSVT 1 [87]. This does not add noise to the query answers, and has no bound on number of True’s to output (i.e., $N$). This is not private for any privacy budget $\epsilon_0$.

**input** : $Q$: queries to the database, $\epsilon_0$: privacy budget
$T$: threshold, $\Delta$: sensitivity

1 function iSVT1($Q$, $T$, $\epsilon_0$, $\Delta$):
2   $out \leftarrow []$
3   $\eta_1 \leftarrow \text{Lap}(2 * \Delta/\epsilon_0)$
4   $\tilde{T} \leftarrow T + \eta_1$
5   // no bounds on number of outputs
6   foreach $q$ in $Q$ do
7     // adds no noise to query answers
8        $\eta_2 \leftarrow 0$
9     if $q + \eta_2 \geq \tilde{T}$ then
10        $out \leftarrow True :: out$
11     else
12        $out \leftarrow False :: out$
13   return ($out$)

This expectation is consistent with the test result shown in Figure 4-4b: the p-value never rises at any test $\epsilon$. This result strongly indicates that this implementation with claimed privacy budget 0.2, 0.7, 1.5 is not private for at least any $\epsilon \leq 2.2$.

iSVT 2 [88]

Another incorrect variant (Algorithm 13) has no bounds on the number of True’s the algorithm can output. Without the bounds, the algorithm will still output True even if it has exhausted its privacy budget. So this variant is not private for any finite $\epsilon_0$.

Figure 4-4c indicates this implementation with privacy budget $\epsilon_0 = 0.2$ is most likely not private for any $\epsilon \leq 0.5$. When $\epsilon_0 = 0.7$, we have detected counterexamples showing the algorithm is likely not private for any $\epsilon \in (0, 2.1)$. When $\epsilon_0 = 1.5$, we have detected counterexamples showing the algorithm is likely not private for any $\epsilon \in (0, 3.0)$.

iSVT 3 [89]

Another incorrect variant (Algorithm 14) adds noise to queries but the noise doesn’t scale with the bound $N$. The actual privacy budget for this variant is $\frac{1+6N}{4}\epsilon_0$ where $\epsilon_0$ is the input privacy budget.
Algorithm 13: iSVT 2 [88]. This one has no bounds on number of True’s (i.e, $N$) to output. This is not private for any finite privacy budget $\varepsilon_0$.

<table>
<thead>
<tr>
<th>function iSVT2($Q$, $T$, $\varepsilon_0$, $\Delta$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

We note that our tool detects the actual privacy cost, as shown in Figure 4-4d, for this incorrect algorithm. Consider privacy budget $\varepsilon_0 = 0.2$. The corresponding line rises at $0.3$, right before the actual budget $\frac{1+6N}{4} \varepsilon_0 = 0.35$ ($N = 1$), suggesting the precision of our tool. The same happens for $\varepsilon_0 = 0.7$ and $1.5$. The two lines rise at $1.1$ and $2.3$, which are close to but before the actual budget $1.225$ and $2.625$, respectively.

iSVT 4 [90]

Another incorrect variant (Algorithm 15) outputs the actual value of noisy query answer when it is above the noisy threshold.

The interesting part of this algorithm is that, since it outputs heterogeneous list of booleans and values, our event selector chooses $\{9\} \times (-2.4, 2.4)$. This means we choose an event that consists of 9 booleans (in this case, Falses) followed by a number in $(-2.4, 2.4)$. Figure 4-4e shows much noise in it because this one is almost correct in the sense that violations of differential privacy happen with very low probability; thus it is hard to detect its incorrectness. But we can still see that the lines all rise later than the corresponding claimed privacy budget $\varepsilon_0$. Hence, our tool correctly concludes that this algorithm does not satisfy $\varepsilon_0$-differential privacy.
Algorithm 14: iSVT 3 [89]. The noise added to queries doesn’t scale with $N$. The actual privacy cost is $\frac{1+6N}{4} \epsilon_0$.

**input**: $Q$: queries to the database, $\epsilon_0$: privacy budget

$T$: threshold, $N$: bound of outputting True’s

$\Delta$: sensitivity

**function** iSVT3($Q$, $T$, $\epsilon_0$, $\Delta$, $N$):

1. $out \leftarrow []$
2. $\eta_1 \leftarrow \text{Lap}(4 \cdot \Delta / \epsilon_0)$
3. $\tilde{T} \leftarrow T + \eta_1$
4. $count \leftarrow 0$

**foreach** $q$ in $Q$ do

5. // noise added doesn’t scale with N
6. $\eta_2 \leftarrow \text{Lap}(4 \cdot \Delta / (3 \cdot \epsilon_0))$
7. if $q + \eta_2 \geq \tilde{T}$ then
8.     $out \leftarrow \text{True :: out}$
9.     $count \leftarrow count + 1$
10.    if $count \geq N$ then
11.        Break
12. else
13.     $out \leftarrow \text{False :: out}$

14. **return** (out)

The counterexamples found by our tool are listed in Table 4-2.

### 4.4.4 Performance

We performed all experiments on a double Intel® Xeon® E5-2620 v4 @ 2.10GHz CPU machine with 64 GB memory. Our tool is implemented in Anaconda distribution of Python 3 and optimized for running in parallel environment to fully utilize the 32 logical cores of the machine.

For each test $\epsilon$, we set the samples of iteration $n$ to be 500,000 for the hypothesis test and 100,000 for the event selector and query generator. Table 4-3 lists the average time spent on hypothesis test for a specific test $\epsilon$ (i.e., the average time spent on generating one single point in the figures) for each algorithm. The results suggest that it is very efficient to run a test for an algorithm against one privacy cost: all tests finish within 23 seconds.

The time difference between Noisy Max, Histogram and Sparse Vector Technique is due to the nature
Algorithm 15: iSVT 4 [90]. When the noisy query answer is above the threshold, output the actual value of noisy query answer.

```plaintext
input : Q: queries to the database, \( \epsilon_0 \): privacy budget
T: threshold, N: bound of outputting True’s
\( \Delta \): sensitivity

1 function iSVT4(Q, T, \( \epsilon_0 \), \( \Delta \), N):
2     out ← []
3     \( \eta_1 \) ← Lap(2 * \( \Delta \)/\( \epsilon_0 \))
4     \( \hat{T} \) ← T + \( \eta_1 \)
5     count ← 0
6     foreach q in Q do
7         \( \eta_2 \) ← Lap(2 * N * \( \Delta \)/\( \epsilon_0 \))
8         if q + \( \eta_2 \) ≥ \( \hat{T} \) then
9             // output numerical value instead of boolean value
10                out ← (q + \( \eta_2 \)) :: out
11                count ← count + 1
12                if count ≥ N then
13                   Break
14         else
15                out ← False :: out
16     return (out)
```

of the algorithms. For SVT, the parameter N is set to 1, meaning that the algorithm will halt once it hit a True branch. For Noisy Max and Histogram, all noise will be calculated and applied to each query answer, consuming more time to calculate p-values. Another factor that will also influence the test time is the search space of events. Correct Noisy Max returns an index which we would have a search space of only integers ranging from 1 to the length of queries. However, the incorrect Noisy Max will return a real number so the search space would be much larger than the correct one, thus taking more time to find a suitable event \( E \). This also occurs in Sparse Vector Technique.
(a) A correct implementation of SVT [11].

(b) iSVT 1 [87] adds no noise to query and threshold.

(c) iSVT 2 [88] no bounds on outputting True's.

(d) iSVT 3 [89] query noise does not scale with $N$.

(e) iSVT 4 [90] outputs the actual query answer when it is above the threshold.

Figure 4-4: Results for variants of Sparse Vector Technique
Table 4-2: Counterexamples Detected for Incorrect Privacy Mechanisms

<table>
<thead>
<tr>
<th>Mechanism (ε₀ = 1.5)</th>
<th>Event E</th>
<th>D1</th>
<th>D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect Noisy Max with Exponential Noise</td>
<td>ω ∈ (−∞, 1.0)</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[0, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>Incorrect Histogram [1]</td>
<td>ω[0] ∈ (−∞, 1.0)</td>
<td>[1, 1, 1, 1, 1]</td>
<td>[2, 1, 1, 1, 1]</td>
</tr>
<tr>
<td>iSVT 1 [87]</td>
<td>t(ω) = 0</td>
<td>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]</td>
<td>[0, 0, 0, 0, 2, 2, 2, 2, 2]</td>
</tr>
<tr>
<td>iSVT 2 [88]</td>
<td>t(ω) = 9</td>
<td>[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 1, 1, 1, 1]</td>
</tr>
<tr>
<td>iSVT 3 [89]</td>
<td>t(ω) = 0</td>
<td>[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]</td>
<td>[0, 0, 0, 0, 1, 1, 1, 1, 1]</td>
</tr>
<tr>
<td>iSVT 4 [90]</td>
<td>(ω.count(False), ω[9]) ∈ (9, -2.4, 2.4)</td>
<td>[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]</td>
<td>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]</td>
</tr>
</tbody>
</table>

Table 4-3: Running Time for Testing Privacy Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Time / Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct Laplace Noisy Max [91]</td>
<td>4.32</td>
</tr>
<tr>
<td>Incorrect Laplace Noisy Max</td>
<td>9.49</td>
</tr>
<tr>
<td>Correct Exponential Noisy Max [91]</td>
<td>4.25</td>
</tr>
<tr>
<td>Incorrect Exponential Noisy Max</td>
<td>8.70</td>
</tr>
<tr>
<td>Histogram [80]</td>
<td>10.39</td>
</tr>
<tr>
<td>Incorrect Histogram</td>
<td>11.28</td>
</tr>
<tr>
<td>iSVT 1 [87]</td>
<td>1.62</td>
</tr>
<tr>
<td>iSVT 2 [88]</td>
<td>4.56</td>
</tr>
<tr>
<td>iSVT 3 [89]</td>
<td>2.56</td>
</tr>
<tr>
<td>iSVT 4 [90]</td>
<td>22.97</td>
</tr>
</tbody>
</table>
Chapter 5

Extra Information Release from Private Selection Mechanisms

5.1 Introduction

The accuracy of differentially private data releases is very important in many applications. One way to improve accuracy is to increase the value of the privacy parameter $\epsilon$, known as the privacy loss budget, as it provides a trade-off between an algorithm’s utility and its privacy protections. However, values of $\epsilon$ that are deemed too high can subject a company to criticisms of not providing enough privacy [13]. For this reason, researchers invest significant effort in tuning algorithms [14, 15, 16, 17, 18, 19] and privacy analyses [20, 21, 18, 22] to provide better utility at the same privacy cost.

Differentially private algorithms are built on smaller components called mechanisms [32]. Popular mechanisms include the Laplace Mechanism [1], Geometric Mechanism [68], Noisy Max [24], Sparse Vector Technique (SVT) [24, 11], and the Exponential Mechanism [25]. As we will explain in this chapter, some of these mechanisms, such as the Exponential Mechanism, Noisy Max and SVT, inadvertently throw away information that is useful for designing accurate algorithms. Our contribution is to present novel variants of these mechanisms that provide more functionality at the same privacy cost (under pure differential privacy).

Given a set of queries, Noisy Max returns the identity (not value) of the query that is likely to have the largest value – it adds noise to each query answer and returns the index of the query with the largest noisy value. The Exponential Mechanism is a replacement for Noisy Max in situations where query answers have utility scores. Meanwhile, SVT is an online algorithm that takes a stream of queries and a predefined public threshold $T$. It tries to return the identities (not values) of the first $k$ queries that are likely larger than the threshold. To do so, it adds noise to the threshold. Then, as it sequentially processes each query, it outputs “\(\top\)” or “\(\bot\)”, depending on whether the noisy value of the current query is larger or smaller than the noisy threshold. The mechanism terminates after $k$ “\(\top\)” outputs.
In recent work [67], using program verification tools, Wang et al. showed that SVT can provide additional information at no additional cost to privacy. That is, when SVT returns “⊤” for a query, it can also return the gap between its noisy value and the noisy threshold.\(^1\) We refer to their algorithm as SVT with Gap.

Inspired by this program verification work, we propose novel variations of Exponential Mechanism, SVT and Noisy Max that add new functionality. For SVT, we show that in addition to releasing this gap information, even stronger improvements are possible – we present an adaptive version that can answer more queries than before by controlling how much privacy budget it uses to answer each query. The intuition is that we would like to spend less of our privacy budget for queries that are probably much larger than the threshold (compared to queries that are probably closer to the threshold). A careful accounting of the privacy impact shows that this is possible. Our experiments confirm that Adaptive SVT with Gap can answer many more queries than the prior versions [11, 24, 67] at the same privacy cost.

For Noisy Max, we show that it too inadvertently throws away information. Specifically, at no additional cost to privacy, it can release an estimate of the gap between the largest and second largest queries (we call the resulting mechanism Noisy Max with Gap). We generalize this result to Noisy Top-K – showing that one can release an estimate of the identities of the \(k\) largest queries and, at no extra privacy cost, release noisy estimates of the pairwise gaps (differences) among the top \(k + 1\) queries.

For Exponential Mechanism, we show that there is also a concept of a gap, which can be used to test whether a non-optimal query was returned. One of the challenges with the Exponential Mechanism is that for efficiency purposes it can use complex sampling algorithms to select the chosen candidate. We show that it is possible to release the noisy gap information even if the sampling algorithms are treated as black boxes (i.e., without access to its intermediate computations).

The extra noisy gap information opens up new directions in the construction of differentially private algorithms and can be used to improve accuracy of certain subsequent queries. For instance, one common task is to use Noisy Max to select the approximate top \(k\) queries and then use additional privacy loss budget to obtain noisy answers to these queries. We show that a postprocessing step can combine these noisy answers with gap information to improve accuracy by up to 66% for counting queries. We provide similar applications for the free gap information in SVT.

\(^1\)This was a surprising result given the number of incorrect attempts at improving SVT based on flawed manual proofs [11] and shows the power of automated program verification techniques.
We prove most of our results using the alignment of random variables framework [11, 14, 67, 43], which is based on the following question: if we change the input to a program, how must we change its random variables so that output remains the same? This technique is used to prove the correctness of almost all pure differential privacy mechanisms [24] but needs to be used in sophisticated ways to prove the correctness of the more advanced algorithms [11, 14, 24, 67, 43]. Nevertheless, alignment of random variables is often used incorrectly (as discussed by Lyu et al. [11]). Thus a secondary contribution of our work is to lay out the precise steps and conditions that must be checked and to provide helpful lemmas that ensure these conditions are met. The Exponential Mechanism does not fit in this framework and requires its own proof techniques, which we explain in Section 5.7. To summarize, our contributions are as follows:

1. We provide a simplified template for writing correctness proofs for intricate differentially private algorithms.

2. Using this technique, we propose and prove the correctness of two new mechanisms: Noisy Top-K with Gap and Adaptive SVT with Gap. These algorithms improve on the original versions of Noisy Max and SVT by taking advantage of free information (i.e., information that can be released at no additional privacy cost) that those algorithms inadvertently throw away. We also show that the free gap information can be maintained even when these algorithms use one-sided noise. This variation improves the accuracy of the gap information.

3. We demonstrate some of the uses of the gap information that is provided by these new mechanisms. When an algorithm needs to use Noisy Max or SVT to select some queries and then measure them (i.e., obtain their noisy answers), we show how the gap information from our new mechanisms can be used to improve the accuracy of the noisy measurements. We also show how the gap information in SVT can be used to estimate the confidence that a query’s true answer really is larger than the threshold.

4. We show that the Exponential Mechanism can also release free gap information. Noting that the free gap extensions of Noisy Max and SVT required access to the internal state of those algorithms, we show that this is unnecessary for Exponential Mechanism. This is useful because implementations of Exponential Mechanism can be very complex and use a variety of different sampling routines.
5. We propose two novel hybridizations of Noisy Max and SVT. These algorithms can release the identities of the approximate top-$k$ queries as long as they are larger than a pre-specified threshold. If fewer than $k$ queries are returned, the algorithms save privacy budget and the gap information they release directly turns into estimates of the query answers (i.e., the algorithm returns the query identities and their answers for free). If $k$ queries are returned then the algorithms still return the gaps between their answers.

6. We empirically evaluate the mechanisms on a variety of datasets to demonstrate their improved utility.

We present background and notation in Section 5.2. We present simplified proof templates for randomness alignment in Section 5.3. We present Adaptive SVT with Gap in Section 5.4 and Noisy Top-K with Gap in Section 5.5. We present the novel algorithms that combine elements of Noisy Max and SVT in 5.6. We present Exponential Mechanism with Gap algorithms in Section 5.7. We present experiments in Section 5.8 and proofs underlying the alignment of randomness framework in Section 5.9.

### 5.2 Background and Notation

In this chapter, we use the following notation. $D$ and $D'$ refer to databases. We use the notation $D \sim D'$ to represent adjacent databases. $M$ denotes a randomized algorithm whose input is a database. $\Omega$ denotes the range of $M$ and $\omega \in \Omega$ denotes a specific output of $M$. We use $E \subseteq \Omega$ to denote a set of possible outputs. Because $M$ is randomized, it also relies on a random noise vector $H \in \mathbb{R}^\infty$. This noise sequence is infinite, but of course $M$ will only use a finite-length prefix of $H$. Some of the commonly used noise distributions for this vector $H$ include the Laplace distribution, the Exponential distribution and the Geometric distribution. Their properties are summarized in Table 5-1.

When we need to draw attention to the noise, we use the notation $M(D, H)$ to indicate the execution of $M$ with database $D$ and randomness coming from $H$. Otherwise we use the notation $M(D)$. We define $\mathcal{H}_D^M = \{H \mid M(D, H) \in E\}$ to be the set of noise vectors that allow $M$, on input $D$, to produce an output in the set $E \subseteq \Omega$. To avoid overburdening the notation, we write $\mathcal{H}_{D:E}$ for $\mathcal{H}_D^M$ and $\mathcal{H}_{D':E}$ for

---

2The notion of adjacency depends on the application. Some papers define it as $D$ can be obtained from $D'$ by modifying one record [1] or by adding/deleting one record [80].
Table 5-1: Commonly Used Noise Distributions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Support</th>
<th>Density/Mass</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lap(β)</td>
<td>ℜ</td>
<td>( \frac{1}{2\beta} \exp\left(-\frac{</td>
<td>x</td>
<td>}{\beta}\right) )</td>
</tr>
<tr>
<td>Exp(β)</td>
<td>[0, ∞)</td>
<td>( \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right) )</td>
<td>( \beta )</td>
<td>( \beta^2 )</td>
</tr>
<tr>
<td>Geo(p)</td>
<td>{0, 1, ...}</td>
<td>( p(1-p)^n )</td>
<td>( \frac{1}{p} )</td>
<td>( \frac{1-p}{p^2} )</td>
</tr>
</tbody>
</table>

\( \mathcal{H}_{D:E}^M \) when \( M \) is clear from the context. When \( E \) consists of a single point \( \omega \), we write these sets as \( \mathcal{H}_{D,\omega} \) and \( \mathcal{H}_{D',\omega} \). This notation is summarized in Table 5-2.

Table 5-2: Notation for Randomness Alignment

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M )</td>
<td>randomized algorithm</td>
</tr>
<tr>
<td>( D, D' )</td>
<td>database</td>
</tr>
<tr>
<td>( D \sim D' )</td>
<td>( D ) is adjacent to ( D' )</td>
</tr>
<tr>
<td>( H = (\eta_1, \eta_2, \ldots) )</td>
<td>input noise vector</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>the space of all output of ( M )</td>
</tr>
<tr>
<td>( \omega )</td>
<td>a possible output; ( \omega \in \Omega )</td>
</tr>
<tr>
<td>( E )</td>
<td>a set of possible outputs; ( E \subseteq \Omega )</td>
</tr>
<tr>
<td>( \mathcal{H}<em>{D:E} = \mathcal{H}</em>{D:E}^M )</td>
<td>( {H \mid M(D, H) \in E} )</td>
</tr>
<tr>
<td>( \mathcal{H}<em>{D:\omega} = \mathcal{H}</em>{D:\omega}^M )</td>
<td>( {H \mid M(D, H) = \omega} )</td>
</tr>
</tbody>
</table>

5.3 Randomness Alignment

To establish that the algorithms we propose are differentially private, we use an idea called randomness alignment that previously had been used to prove the privacy of a variety of sophisticated algorithms [24, 11, 14] and incorporated into verification/synthesis tools [43, 67, 42]. While powerful, this technique is also easy to use incorrectly [11], as there are many technical conditions that need to be checked. In this section, we present results (namely Lemma 5.1) that significantly simplify this process and make it easy to prove the correctness of our proposed algorithms.

In general, to prove \( \epsilon \)-differential privacy for an algorithm \( M \), one needs to show \( P(M(D, H) \in E) \leq \epsilon P(M(D', H') \in E) \) for all pairs of adjacent databases \( D \sim D' \) and sets of possible outputs \( E \subseteq \Omega \). In our notation, this inequality is represented as \( P(\mathcal{H}_{D:E}) \leq \epsilon P(\mathcal{H}_{D':E}) \). Establishing such
inequalities is often done with the help of a function $\phi_{D,D'}$, called a randomness alignment (there is a function $\phi_{D,D'}$ for every pair $D \sim D'$), that maps noise vectors $H$ into noise vectors $H'$ so that $M(D',H')$ produces the same output as $M(D,H)$. Formally,

**Definition 5.1** (Randomness Alignment). Let $M$ be a randomized algorithm. Let $D \sim D'$ be a pair of adjacent databases. A randomness alignment is a function $\phi_{D,D'} : \mathbb{R}^\infty \to \mathbb{R}^\infty$ such that

1. The alignment does not output invalid noise vectors (e.g., it cannot produce negative numbers for random variables that should have the exponential distribution).

2. For all $H$ on which $M(D,H)$ terminates, $M(D,H) = M(D',\phi_{D,D'}(H))$.

**Example 1.** Let $D$ be a database that records the salary of every person, which is guaranteed to be between 0 and 100. Let $q(D)$ be the sum of the salaries in $D$. The sensitivity of $q$ is thus 100. Let $H = (\eta_1, \eta_2, \ldots)$ be a vector of independent $\text{Lap}(100/\epsilon)$ random variables. The Laplace mechanism outputs $q(D) + \eta_1$ (and ignores the remaining variables in $H$). For every pair of adjacent databases $D \sim D'$, one can define the corresponding randomness alignment $\phi_{D,D'}(H) = H' = (\eta'_1, \eta'_2, \ldots)$, where $\eta'_1 = \eta_1 + q(D) - q(D')$ and $\eta'_i = \eta_i$ for $i > 1$. Note that $q(D) + \eta_1 = q(D') + \eta'_1$, so the output of $M$ remains the same.

In practice, $\phi_{D,D'}$ is constructed locally (piece by piece) as follows. For each possible output $\omega \in \Omega$, one defines a function $\phi_{D,D',\omega}$ that maps noise vectors $H$ into noise vectors $H'$ with the following properties: if $M(D,H) = \omega$ then $M(D',H') = \omega$ (that is, $\phi_{D,D',\omega}$ only cares about what it takes to produce the specific output $\omega$). We obtain our randomness alignment $\phi_{D,D'}$ in the obvious way by piecing together the $\phi_{D,D',\omega}$ as follows: $\phi_{D,D'}(H) = \phi_{D,D',\omega^*}(H)$, where $\omega^*$ is the output of $M(D,H)$. Formally,

**Definition 5.2** (Local Alignment). Let $M$ be a randomized algorithm. Let $D \sim D'$ be a pair of adjacent databases and $\omega$ a possible output of $M$. A local alignment for $M$ is a function $\phi_{D,D',\omega} : \mathcal{H}_{D,\omega} \to \mathcal{H}_{D',\omega}$ (see notation in Table 5-2) such that for all $H \in \mathcal{H}_{D,\omega}$, we have $M(D,H) = M(D',\phi_{D,D',\omega}(H))$.

**Example 2.** Continuing the setup from Example 1, consider the mechanism $M_1$ that, on input $D$, outputs $\top$ if $q(D) + \eta_1 \geq 10,000$ (i.e. if the noisy total salary is at least 10,000) and $\bot$ if $q(D) + \eta_1 < 10,000$. Let $D'$ be a database that differs from $D$ in the presence/absence of one record. Consider the local alignments
\( \phi_{D,D',\tau} \) and \( \phi_{D,D',\perp} \) defined as follows. \( \phi_{D,D',\tau}(H) = H' = (\eta'_1, \eta'_2, \ldots) \) where \( \eta'_1 = \eta_1 + 100 \) and \( \eta'_i = \eta_i \) for \( i > 1 \); and \( \phi_{D,D',\perp}(H) = H'' = (\eta''_1, \eta''_2, \ldots) \) where \( \eta''_1 = \eta_1 - 100 \) and \( \eta''_i = \eta_i \) for \( i > 1 \). Clearly, if \( M_1(D, H) = \top \) then \( M_1(D', H') = \top \) and if \( M_1(D, H) = \perp \) then \( M_1(D', H'') = \perp \). We piece these two local alignments together to create a randomness alignment \( \phi_{D,D'}(H) = H' = (\eta'_1, \eta'_2, \ldots) \) where:

\[
\eta'_1 = \begin{cases} 
\eta_1 + 100 & \text{if } M(D, H) = \top \text{ (i.e. } q(D) + \eta_1 \geq 10,000) \\
\eta_1 - 100 & \text{if } M(D, H) = \perp \text{ (i.e. } q(D) + \eta_1 < 10,000) 
\end{cases}
\]

\( \eta'_i = \eta_i \) for \( i > 1 \).

**Special properties of alignments.** Not all alignments can be used to prove differential privacy. In this section we discuss some additional properties that help prove differential privacy. We first make two mild assumptions about the mechanism \( M \): (1) it terminates with probability\(^3\) one and (2) based on the output of \( M \), we can determine how many random variables it used. The vast majority of differentially private algorithms in the literature satisfy these properties.

We next define two properties of a local alignment: whether it is acyclic and what its cost is.

**Definition 5.3 (Acyclic).** Let \( M \) be a randomized algorithm. Let \( \phi_{D,D',\omega} \) be a local alignment for \( M \). For any \( H = (\eta_1, \eta_2, \ldots) \), let \( H' = (\eta'_1, \eta'_2, \ldots) \) denote \( \phi_{D,D',\omega}(H) \). We say that \( \phi_{D,D',\omega} \) is acyclic if there exists a permutation \( \pi \) and piecewise differentiable functions \( \psi_{D,D',\omega}^{(j)} \) such that:

\[
\eta'_{\pi(1)} = \eta_{\pi(1)} + \text{constant that only depends on } D, D', \omega \\
\eta'_{\pi(j)} = \eta_{\pi(j)} + \psi_{D,D',\omega}^{(j)}(\eta_{\pi(1)}, \ldots, \eta_{\pi(j-1)}) \text{ for } j \geq 2
\]

Essentially, a local alignment \( \phi_{D,D',\omega} \) is acyclic if there is some ordering of the variables so that \( \eta'_j \) is the sum of \( \eta_j \) and a function of the variables that came earlier in the ordering. The local alignments \( \phi_{D,D',\tau} \) and \( \phi_{D,D',\perp} \) from Example 2 are both acyclic (in general, each local alignment function is allowed to have its own specific ordering and differentiable functions \( \psi_{D,D',\omega}^{(j)} \)). The pieced-together

\(^3\)That is, for each input \( D \), there might be some random vectors \( H \) for which \( M \) does not terminate, but the total probability of these vectors is 0, so we can ignore them.
randomness alignment $\phi_{D,D'}$ itself need not be acyclic.

**Definition 5.4 (Alignment Cost).** Let $M$ be a randomized algorithm that uses $H$ as its source of randomness. Let $\phi_{D,D',\omega}$ be a local alignment for $M$. For any $H = (\eta_1, \eta_2, \ldots)$, let $H' = (\eta'_1, \eta'_2, \ldots)$ denote $\phi_{D,D',\omega}(H)$. Suppose each $\eta_i$ is generated independently from a distribution $f_i$ with the property that, for some $\alpha_i$, $\ln \frac{f_i(x)}{f_i(y)} \leq \frac{|x-y|}{\alpha_i}$ for all $x, y$ in the domain of $f_i$. Then the cost of $\phi_{D,D',\omega}$ is defined as:

$$\text{cost}(\phi_{D,D',\omega}) = \sum_{i=0}^{\infty} \frac{1}{100} |\eta'_i - \eta_i| = \sum_{i=0}^{\infty} \frac{1}{100} |q(D') - q(D)| \leq \epsilon.$$  

Distributions that we use in this paper (see Table 5-1) with this property include the Laplace $(\text{i.e., Lap}(\alpha_i))$, Exponential $(\text{i.e., Exp}(\alpha_i))$, and Geometric $(\text{i.e., Geo}(1 - e^{-1/\alpha_i}))$.

The following lemma uses those properties to establish that $M$ satisfies $\epsilon$-differential privacy.

**Lemma 5.1.** Let $M$ be a randomized algorithm with input randomness $H = (\eta_1, \eta_2, \ldots)$. If the following conditions are satisfied, then $M$ satisfies $\epsilon$-differential privacy.

1. $M$ terminates with probability 1.
2. The number of random variables used by $M$ can be determined from its output.
3. Each $\eta_i$ is generated independently from a distribution $f_i$ with the property that $\ln \frac{f_i(x)}{f_i(y)} \leq \frac{|x-y|}{\alpha_i}$ for all $x, y$ in the domain of $f_i$.
4. For every $D \sim D'$ and $\omega$ there exists a local alignment $\phi_{D,D',\omega}$ that is acyclic with $\text{cost}(\phi_{D,D',\omega}) \leq \epsilon$.
5. For each $D \sim D'$ the number of distinct local alignments is countable. That is, the set $\{\phi_{D,D',\omega} : \omega \in \Omega\}$ is countable (i.e., for many choices of $\omega$ we get the same exact alignment function).

We defer the proof to Section 5.9.

**Example 3.** Consider the randomness alignment $\phi_{D,D'}$ from Example 1. We can define all of the local alignments $\phi_{D,D',\omega}$ to be the same function: $\phi_{D,D',\omega}(H) = \phi_{D,D'}(H)$. Clearly $\text{cost}(\phi_{D,D',\omega}) = \sum_{i=0}^{\infty} \frac{1}{100} |\eta'_i - \eta_i| = \frac{1}{100} |q(D') - q(D)| \leq \epsilon$. For Example 2, there are two acyclic local alignments $\phi_{D,D',\top}$ and $\phi_{D,D',\perp}$, both have cost $= 100 \cdot \frac{\epsilon}{100} = \epsilon$. The other conditions in Lemma 5.1 are trivial to check. Thus both mechanisms satisfy $\epsilon$-differential privacy by Lemma 5.1.
5.4 Improving Sparse Vector

In this section we propose an adaptive variant of SVT that can answer more queries than both the original SVT [24, 11] and the SVT with Gap of Wang et al. [67]. We explain how to tune its privacy budget allocation. We further show that using other types of random noise, such as exponential and geometric random variables, in place of the Laplace, makes the free gap information more accurate at the same cost to privacy. Finally, we discuss how the free gap information can be used for improved utility of data analysis.

5.4.1 Adaptive SVT with Gap

The Sparse Vector Technique (SVT) is designed to solve the following problem in a privacy-preserving way: given a stream of queries (with sensitivity 1), find the first \( k \) queries whose answers are larger than a public threshold \( T \). This is done by adding noise to the queries and threshold and finding the first \( k \) queries whose noisy answers exceed the noisy threshold. Sometimes this procedure creates a feeling of regret – if these \( k \) queries are much larger than the threshold, we could have used more noise (hence consumed less privacy budget) to achieve the same result. In this section, we show that Sparse Vector can be made adaptive – so that it will probably use more noise (less privacy budget) for the larger queries. This means if the first \( k \) queries are very large, it will still have privacy budget left over to find additional queries that are likely to be over the threshold. Adaptive SVT is shown in Algorithm 16.

The main idea behind this algorithm is that, given a target privacy budget \( \epsilon \) and an integer \( k \), the algorithm will create three budget parameters: \( \epsilon_0 \) (budget for the threshold), \( \epsilon_1 \) (baseline budget for each query) and \( \epsilon_2 \) (smaller alternative budget for each query, \( \epsilon_2 < \epsilon_1 \)). The privacy budget allocation between threshold and queries is controlled by a hyperparameter \( \theta \in (0, 1) \) on Line 2. These budget parameters are used as follows. First, the algorithm adds \( \text{Lap}(1/\epsilon_0) \) noise to the threshold and consumes \( \epsilon_0 \) of the privacy budget. Then, when a query comes in, the algorithm first adds a lot of noise (i.e., \( \text{Lap}(2/\epsilon_2) \)) to the query. The first “if” branch checks if this value is much larger than the noisy threshold (i.e., checks if the gap is \( \geq 2\sigma \) for some\(^4 \sigma \)). If so, then it outputs the following three items: \( 1) \ T, 

\(^4\)In our algorithm, we set \( \sigma \) to be the standard deviation of the noise distribution.
Algorithm 16: Adaptive SVT with Gap. The hyperparameter $\theta \in (0, 1)$ controls the budget allocation between threshold and queries.

**input**: $q$: a list of queries of global sensitivity 1

$D$: database, $\epsilon$: privacy budget, $T$: threshold

$k$: minimum number of above-threshold queries algorithm is able to output

```plaintext
function AdaptiveSparse ($q$, $D$, $T$, $k$, $\epsilon$):

$\epsilon_0 \leftarrow \theta \epsilon$; $\epsilon_1 \leftarrow (1 - \theta)\epsilon/k$; $\epsilon_2 \leftarrow \epsilon_1/2$

$\sigma \leftarrow 2\sqrt{2}/\epsilon_2$ // std dev of Lap(2/\epsilon_2)

$\eta \leftarrow \text{Lap}(1/\epsilon_0)$; $\bar{T} \leftarrow T + \eta$

cost $\leftarrow \epsilon_0$

foreach $i \in \{1, \cdots, \text{len}(q)\}$ do

$\xi_i \leftarrow \text{Lap}(2/\epsilon_2)$; $\tilde{q}_i \leftarrow q_i(D) + \xi_i$

$\eta_i \leftarrow \text{Lap}(2/\epsilon_1)$; $\hat{q}_i \leftarrow q_i(D) + \eta_i$

if $\tilde{q}_i - \bar{T} \geq 2\sigma$ then

output: ($\top$, $\tilde{q}_i - \bar{T}$, bud_used $= \epsilon_2$)

cost $\leftarrow$ cost $+ \epsilon_2$

else if $\hat{q}_i - \bar{T} \geq 0$ then

output: ($\top$, $\hat{q}_i - \bar{T}$, bud_used $= \epsilon_1$)

cost $\leftarrow$ cost $+ \epsilon_1$

else

output: ($\bot$, bud_used $= \emptyset$)

if cost $> \epsilon - \epsilon_1$ then break
```

(2) the noisy gap, and (3) the amount of privacy budget used for this query (which is $\epsilon_2$). The use of alignments will show that failing this “if” branch consumes no privacy budget. If the first “if” branch fails, then the algorithm adds more moderate noise (i.e., Lap(2/$\epsilon_1$)) to the query answer. If this noisy value is larger than the noisy threshold, the algorithm outputs: (1’) $\top$, (2’) the noisy gap, and (3’) the amount of privacy budget consumed (i.e., $\epsilon_1$). If this “if” condition also fails, then the algorithm outputs: (1″) $\bot$ and (2″) the privacy budget consumed (0 in this case).

To summarize, there is a one-time cost for adding noise to the threshold. Then, for each query, if the top branch succeeds the privacy budget consumed is $\epsilon_2$, if the middle branch succeeds, the privacy cost is $\epsilon_1$, and if the bottom branch succeeds, there is no additional privacy cost. These properties can be easily seen by focusing on the local alignment – if $M(D, H)$ produces a certain output, how much does $H$ need to change to get a noise vector $H'$ so that $M(D', H')$ returns the same exact output.
Local alignment. To create a local alignment for each pair $D \sim D'$, let $H = (\eta, \xi_1, \eta_1, \xi_2, \eta_2, \ldots)$ where $\eta$ is the noise added to the threshold $T$, and $\xi_i$ (resp. $\eta_i$) is the noise that should be added to the $i^{\text{th}}$ query $q_i$ in Line 7 (resp. Line 8), if execution ever reaches that point. We view the output $\omega = (w_1, \ldots, w_s)$ as a variable-length sequence where each $w_i$ is either $\perp$ or a nonnegative gap (we omit the $\top$ as it is redundant), together with a tag $\in \{0, e_1, e_2\}$ indicating which branch $w_i$ is from (and the privacy budget consumed to output $w_i$). Let $I_\omega = \{i \mid \text{tag}(w_i) = e_2\}$ and $J_\omega = \{i \mid \text{tag}(w_i) = e_1\}$. That is, $I_\omega$ is the set of indexes where the output is a gap from the top branch, and $J_\omega$ is the set of indexes where the output is a gap from the middle branch. For $H \in \mathcal{H}_{D,\omega}$ define $\phi_{D,D',\omega}(H) = H' = (\eta', \xi'_1, \eta'_1, \xi'_2, \eta'_2, \ldots)$ where

$$
\eta' = \eta + 1,
$$

$$
(\xi'_i, \eta'_i) = \begin{cases} 
(\xi_i + 1 + q_i - q'_i, \eta_i), & i \in I_\omega \\
(\xi_i, \eta_i + 1 + q_i - q'_i), & i \in J_\omega \\
(\xi_i, \eta_i), & \text{otherwise}
\end{cases}
$$

In other words, we add 1 to the noise that was added to the threshold (thus if the noisy $q(D)$ failed a specific branch, the noisy $q(D')$ will continue to fail it because of the higher noisy threshold). If a noisy $q(D)$ succeeded in a specific branch, we adjust the query’s noise so that the noisy version of $q(D')$ will succeed in that same branch.

Lemma 5.2. Let $M$ be the Adaptive SVT with Gap algorithm. For all $D \sim D'$ and $\omega$, the functions $\phi_{D,D',\omega}$ defined above are acyclic local alignments for $M$. Furthermore, for every pair $D \sim D'$, there are countably many distinct $\phi_{D,D',\omega}$.

Proof. Pick an adjacent pair $D \sim D'$ and an $\omega = (w_1, \ldots, w_s)$. For a given $H = (\eta, \xi_1, \eta_1, \ldots)$ such that $M(D, H) = \omega$, let $H' = (\eta', \xi'_1, \eta'_1, \ldots) = \phi_{D,D',\omega}(H)$. Suppose $M(D', H') = \omega' = (w'_1, \ldots, w'_t)$. Our goal is to show $\omega' = \omega$. Choose an $i \leq \min(s, t)$.

- If $i \in I_\omega$, then by (5.1) we have

$$
q'_i + \xi'_i - (T + \eta') = q'_i + \xi'_i + 1 + q_i - q'_i - (T + \eta + 1)
$$
= q_i + \xi_i - (T + \eta) \geq \sigma.

This means the first “if” branch succeeds in both executions and the gaps are the same. Therefore, 
\( w'_i = w_i. \)

- If \( i \in \mathcal{J}_\omega, \) then by (5.1) we have
\
\begin{align*}
q'_i + \xi'_i - (T + \eta') & = q'_i + \xi_i - (T + \eta + 1) \\
& = q'_i - 1 + \xi_i - (T + \eta) \leq q_i + \xi_i - (T + \eta) < \sigma, \\
q'_i + \eta'_i - (T + \eta') & = q'_i + \eta_i + 1 + q_i - q'_i - (T + \eta + 1) \\
& = q_i + \eta_i - (T + \eta) \geq 0.
\end{align*}
\)

The first inequality is due to the sensitivity restriction: \( |q_i - q'_i| \leq 1 \implies q'_i - 1 \leq q_i. \) These two equations mean that the first “if” branch fails and the second “if” branch succeeds in both executions, and the gaps are the same. Hence \( w'_i = w_i. \)

- If \( i \not\in \mathcal{I}_\omega \cup \mathcal{J}_\omega, \) then by a similar argument we have
\
\begin{align*}
q'_i + \xi'_i - (T + \eta') & \leq q_i + \xi_i - (T + \eta) < \sigma, \\
q'_i + \eta'_i - (T + \eta') & \leq q_i + \eta_i - (T + \eta) < 0.
\end{align*}
\)

Hence both executions go to the last “else” branch and \( w'_i = (\perp, 0) = w_i. \)

Therefore for all \( 1 \leq i \leq \min(s, t), \) we have \( w'_i = w_i. \) That is, either \( \omega' \) is a prefix of \( \omega, \) or vice versa. Let \( q \) be the vector of queries passed to the algorithm and let \( \text{len}(q) \) be the number of queries it contains (which can be finite or infinity). By the termination condition of Algorithm 16 we have two possibilities.

1. \( s = \text{len}(q): \) in this case there is still enough privacy budget left after answering \( s - 1 \) above-threshold queries, and we must have \( t = \text{len}(q) \) too because \( M(D', H') \) will also run through all the queries (it cannot stop until it has exhausted the privacy budget or hits the end of the query sequence).

2. \( s < \text{len}(q): \) in this case the privacy budget is exhausted after outputting \( w_s \) and we must also have \( t = s. \)
Thus \( t = s \) and hence \( \omega' = \omega \). The local alignments are clearly acyclic (e.g., use the identity permutation).

Note that \( \phi_{D,D',\omega} \) only depends on \( \omega \) through \( I_\omega \) and \( J_\omega \) (the sets of queries whose noisy values were larger than the noisy threshold). There are only countably many possibilities for \( I_\omega \) and \( J_\omega \) and thus countably many distinct \( \phi_{D,D',\omega} \).

\[ \square \]

**Alignment cost and privacy.** Now we establish the alignment cost and the privacy property of Algorithm 16.

**Theorem 5.3.** The Adaptive SVT with Gap satisfies \( \epsilon \)-differential privacy.

**Proof.** First we bound the cost of the alignment function defined by Equation (5.1). We use the \( \epsilon_0, \epsilon_1, \epsilon_2 \) and \( \epsilon \) defined in Algorithm 16. From (5.1) we have

\[
\begin{align*}
\text{cost}(\phi_{D,D',\omega}) &= \epsilon_0 |\eta' - \eta| + \sum_{i=1}^{\infty} \left( \frac{\epsilon_2}{2} |\xi_i' - \xi_i| + \frac{\epsilon_1}{2} |\eta_i' - \eta_i| \right) \\
&= \epsilon_0 + \sum_{i \in I_\omega} \frac{\epsilon_2}{2} |1 + q_i - q_i'| + \sum_{i \in J_\omega} \frac{\epsilon_1}{2} |1 + q_i - q_i'| \\
&\leq \epsilon_0 + \epsilon_2 |I_\omega| + \epsilon_1 |J_\omega| \leq \epsilon.
\end{align*}
\]

The first inequality is from the assumption on sensitivity: \(|1 + q_i - q_i'| \leq 1 + |q_i - q_i'| \leq 2\). The second inequality is from loop invariant on Line 17: \( \epsilon_0 + \epsilon_2 |I_\omega| + \epsilon_1 |J_\omega| = \text{cost} \leq \epsilon - \epsilon_1 + \max(\epsilon_1, \epsilon_2) = \epsilon \).

Conditions 1, 2, 3 of Lemma 5.1 are trivial to check, 4 and 5 follow from Lemma 5.2 and the above bound on cost. Thus Theorem 5.3 follows from Lemma 5.1. \( \square \)

Algorithm 16 can be easily extended with multiple additional “if” branches. For simplicity we do not include such variations. In our setting, \( \epsilon_2 = \epsilon_1/2 \) so, theoretically, if queries are very far from the threshold, our adaptive version of Sparse Vector will be able to find twice as many of them as the non-adaptive version. Lastly, if all queries are monotonic queries, then Algorithm 16 can be further improved: we can use \( \text{Lap}(1/\epsilon_2) \) in Line 7 and \( \text{Lap}(1/\epsilon_1) \) noises in Line 8 instead.\(^5\)

\(^5\)In the case of monotonic queries, if \( \forall i : q_i \geq q_i' \), then the alignment changes slightly: we set \( \eta' = \eta \) (the random variable added to the threshold) and set the adjustment to noise in the winning “if” branches to \( q_i - q_i' \) instead of \( 1 + q_i - q_i' \) (hence cost terms become \( |q_i - q_i'| \) instead of \( |1 + q_i - q_i'| \)). If \( \forall i : q_i \leq q_i' \) then we keep the original alignment but in the cost calculation we note that \( |1 + q_i - q_i'| \leq 1 \) (due to the monotonicity and sensitivity).
Choice of $\theta$. We can optimize the budget allocation between threshold noise and query noises by following the methodology of [11], which is equivalent to minimizing the variance of the gap between a noisy query and the threshold. If the majority of gaps are expected to be returned from the top branch, then we optimize $\text{Var}(\tilde{q} - T) = \frac{2}{\epsilon_0^2} + \frac{8}{\epsilon_1^2} = \frac{2}{\epsilon_0^2}(\frac{1}{\theta^2} + \frac{16k^2}{(1-\theta)^2})$. This variance attains its minimum value of $2(1 + \sqrt{16k^2})/\epsilon^2$ when $\theta = 1/(1 + \sqrt{16k^2})$. If on the other hand the majority of gaps are expected to be returned from the middle branch, then we optimize $\text{Var}(\hat{q} - T) = \frac{2}{\epsilon_0^2} + \frac{8}{\epsilon_1^2} = \frac{2}{\epsilon_0^2}(\frac{1}{\theta^2} + \frac{4k^2}{(1-\theta)^2})$. In this case, the minimum value is $2(1 + \sqrt{4k^2})/\epsilon^2$ when $\theta = 1/(1 + \sqrt{4k^2})$. If all queries are monotone, then the optimal variance further reduces to $2(1 + \sqrt{4k^2})/\epsilon^2$ in the top branch when $\theta = 1/(1 + \sqrt{4k^2})$, and $2(1 + \sqrt{k^2})/\epsilon^2$ in the middle branch when $\theta = 1/(1 + \sqrt{k^2})$.

These allocation strategies also extend to SVT with Gap (originally proposed in [67]). SVT with Gap can be obtained by removing the first branch of Algorithm 16 (Line 9 through 11) or setting $\sigma = \infty$. For reference, we show its pseudocode below as Algorithm 17. In [67], $\theta$ is set to 0.5, which is suboptimal. The optimal value is $\theta = 1/(1 + \sqrt{4k^2})$.

```
Algorithm 17: SVT with Gap [67]

input : same as Algorithm 16

function GapSparse (q, D, T, k, $\epsilon$):
  $\epsilon_0 \leftarrow \theta \epsilon$; $\epsilon_1 \leftarrow (1 - \theta)\epsilon/k$;
  $\eta \leftarrow \text{Lap}(1/\epsilon_0)$; $\bar{T} \leftarrow T + \eta$
  cost $\leftarrow \epsilon_0$
  foreach $i \in \{1, \cdots, \text{len}(q)\}$ do
    $\eta_i \leftarrow \text{Lap}(2/\epsilon_1)$; $\tilde{q}_i \leftarrow q_i(D) + \eta_i$
    if $\tilde{q}_i - \bar{T} \geq 0$ then
      output: ($\top$, $\tilde{q}_i - \bar{T}$, bud_used = $\epsilon_1$)
      cost $\leftarrow$ cost + $\epsilon_1$
    else
      output: ($\bot$, bud_used = $\emptyset$)
  if cost $> \epsilon - \epsilon_1$ then break
```

5.4.2 Using Exponential or Geometric Noise.

In this section, we show that Adaptive SVT with Gap also satisfies differential privacy if the Laplace noise is replaced by the exponential distribution or the geometric distribution (when query answers are
guaranteed to be integers). Both of these are one-sided distributions that result in a gap estimate with lower variance (see Table 5-1 for information about those distributions). The same result carries over to SVT with Gap [67].

**Exponential noise.** When using random noise from the exponential distribution, we need to subtract the expected value of the noise from the queries and threshold. We make the following changes to Lines 3, 4, 7 and 8 of Algorithm 16:

3. $\sigma \leftarrow 2/\epsilon_2$ \hspace{1cm} // std dev of Exp(2/\epsilon_2)
4. $\eta \leftarrow \text{Exp}(1/\epsilon_0)$; $T \leftarrow T + \eta - 1/\epsilon_0$
7. $\xi_i \leftarrow \text{Exp}(2/\epsilon_2)$; $\tilde{q}_i \leftarrow q_i(D) + \xi_i - 2/\epsilon_2$
8. $\eta_i \leftarrow \text{Exp}(2/\epsilon_1)$; $\hat{q}_i \leftarrow q_i(D) + \eta_i - 2/\epsilon_1$

In more detail, the changes are:

1. Line 3: the algorithm changes the value of $\sigma$ from $2\sqrt{2}/\epsilon_2$, the standard deviation of Lap(2/\epsilon_2), to $2/\epsilon_2$, the standard deviation of Exp(2/\epsilon_2). It is worth repeating that the one-sided exponential noise results in a reduction of variance.

2. Lines 4, 7 and 8: change Laplace noises to exponential noises of the same scale, and then subtracts the expected values of the noises.

If all queries are counting queries, we further replace $\epsilon_1$ and $\epsilon_2$ in Line 3, 7 and 8 with $2\epsilon_1$ and $2\epsilon_2$ respectively.

**Geometric noise.** When all queries have integer values (e.g. counting queries), we could utilize geometric noise to ensure that the gap is also an integer. To do so we make the following changes to Algorithm 16:

3. $\sigma \leftarrow e^{\frac{\epsilon_2}{2}}/(e^{\frac{\epsilon_2}{2}}-1)$ \hspace{1cm} // std dev of Geo(1−$e^{-\frac{\epsilon_2}{2}}$)
4. $\eta \leftarrow \text{Geo}(1−e^{-\epsilon_0})$; $T \leftarrow T + \eta - 1/(1−e^{-\epsilon_0})$
7. $\xi_i \leftarrow \text{Geo}(1−e^{-\frac{\epsilon_2}{2}})$; $\tilde{q}_i \leftarrow q_i(D) + \xi_i - 1/(1−e^{-\frac{\epsilon_2}{2}})$
8. $\eta_i \leftarrow \text{Geo}(1−e^{-\frac{\epsilon_1}{2}})$; $\hat{q}_i \leftarrow q_i(D) + \eta_i - 1/(1−e^{-\frac{\epsilon_1}{2}})$
If all queries are counting queries, we further replace $\epsilon_1$ and $\epsilon_2$ in Line 3, 7 and 8 with $2\epsilon_1$ and $2\epsilon_2$ respectively.

**Local alignment and privacy.** The alignment in Equation 5.1 for the Adaptive SVT with Gap with Laplace noise also works for both exponential noise and geometric noise, because $\eta' = \eta = 1$ and $\xi_i' - \xi_i, \eta_i' - \eta_i \in \{0, 1 + q_i - q_i'\}$. The value $1 + q_i - q_i'$ is always $\geq 0$ and is an integer when $q_i, q_i'$ are integers.

Recall that if $f(x)$ is the probability density function of $\text{Exp}(\beta)$, then $\ln \frac{f(x)}{f(y)} \leq \frac{1}{\beta} |x - y|$. Similarly, if $g(x)$ is the probability mass function of $\text{Geo}(p)$, then $\ln \frac{g(x)}{g(y)} = \ln \frac{p(1-p)^y}{p(1-p)^{y+1}} \leq -\ln(1 - p)|x - y|$. Therefore, our choice of the parameters ensures that the alignment cost is the same as that of Laplace noise, which is bounded by $\epsilon$. Thus both variants are $\epsilon$-differentially private.

**Choice of $\theta$.** As before, we choose the $\theta$ that minimizes the variance of the gap to make the result most accurate. Note that exponential distribution has half the variance of the Laplace distribution of the same scale. Thus, when exponential noise is used, the minimum variance of the gap is $(1 + \sqrt{16k^2})^3/e^2$ in the top branch when $\theta = 1/(1 + \sqrt{16k^2})$, and $(1 + \sqrt{4k^2})^3/e^2$ in the middle branch when $\theta = 1/(1 + \sqrt{4k^2})$.

If all queries are monotone, then the optimal variance further reduces to $(1 + \sqrt{4k^2})^3/e^2$ in the top branch when $\theta = 1/(1 + \sqrt{4k^2})$, and $(1 + \sqrt{k^2})^3/e^2$ in the middle branch when $\theta = 1/(1 + \sqrt{k^2})$.

Since the geometric distribution is the discrete analogue of the exponential distribution, the above results apply to geometric noise as well. For example, when all queries are counting queries and geometric noise is used, then $\text{Var}(\hat{q}_i - \overline{T}) = \frac{e^{\theta e_0}}{(e^{\theta e_0} - 1)^2} + \frac{e^{\theta e_1}}{(e^{\theta e_1} - 1)^2} + \frac{e^{(1-\theta)e/k}}{(e^{(1-\theta)e/k} - 1)^2}$ in the middle branch. The variance of the gap, albeit complicated, is a convex function of $\theta$ on $(0, 1)$. We used the LBFGS algorithm [92] from SciPy to find the $\theta$ where the variance is minimum, and found that those values are almost the same as those for exponential noise (See Fig. 5-1). Therefore, we can use the budget allocation strategy for exponential noise as the strategy for geometric noise too.
Figure 5-1: The blue dots are values of $\theta_{\text{min}} = \arg\min (\frac{\theta e^{\theta e}}{(e^{\theta e} - 1)^2} + \frac{e^{(1-\theta)e/k}}{(e^{(1-\theta)e/k} - 1)^2})$ for $k$ from 1 to 50. The orange curve is the function $\theta = \frac{1}{1 + \sqrt{k^2}}$.

5.4.3 Utilizing Gap Information

When SVT with Gap or Adaptive SVT with Gap returns a gap $\gamma_i$ for a query $q_i$, we can add to it the public threshold $T$. This means $\gamma_i + T$ is an estimate of the value of $q_i(D)$. We can ask two questions: how can we improve the accuracy of this estimate and how can we be confident that the true answer $q_i(D)$ is really larger than the threshold $T$?

Lower confidence interval. Recall that the randomness in the gap in Adaptive SVT with Gap (Algorithm 16) is of the form $\eta_i - \eta$ where $\eta$ and $\eta_i$ are independent zero mean Laplace variables with scale $1/\epsilon_0$ and $1/\epsilon$, where $\epsilon$ is either $\epsilon_1$ or $\epsilon_2$, depending on the branch. The random variable $\eta_i - \eta$ has the following lower tail bound:

**Lemma 5.4.** For any $t \geq 0$ we have

$$
P(\eta_i - \eta \geq -t) = \begin{cases} 
1 - \frac{e^{\epsilon_0 t} - e^{\epsilon_0 t}}{2(\epsilon_0^2 - \epsilon^2)} & \epsilon_0 \neq \epsilon \\
1 - \frac{(2\epsilon_0 t) e^{-\epsilon_0 t}}{4} & \epsilon_0 = \epsilon
\end{cases}
$$

For proof see Section 5.10. For any confidence level, say 95%, we can use this result to find a number $t_{95}$ such that $P((\eta_i - \eta) \geq -t_{95}) = .95$. This is a lower confidence bound, so that the true value $q_i(D)$ is $\geq$ our estimated value $\gamma_i + T$ minus $t_{95}$ with probability 0.95.
Improving accuracy. To improve accuracy, one can split the privacy budget $\epsilon$ in half. The first half $\epsilon' \equiv \epsilon/2$ can be used to run SVT with Gap (or Adaptive SVT with Gap) and the second half $\epsilon'' \equiv \epsilon/2$ can be used to provide an independent noisy measurement of the selected queries (i.e., if we selected $k$ queries, we add $\operatorname{Lap}(k/\epsilon'')$ noise to each one). Denote the $k$ selected queries by $q_1, \ldots, q_k$, the noisy gaps by $\gamma_1, \ldots, \gamma_k$ and the independent noisy measurements by $\alpha_1, \ldots, \alpha_k$. The noisy estimates can be combined together with the gaps to get improved estimates $\beta_i$ of $q_i(D)$ in the standard way (inverse-weighting by variance):

$$\beta_i = \left( \frac{\alpha_i}{\operatorname{Var}(\alpha_i)} + \frac{\gamma_i + T}{\operatorname{Var}(\gamma_i)} \right) / \left( \frac{1}{\operatorname{Var}(\alpha_i)} + \frac{1}{\operatorname{Var}(\gamma_i)} \right).$$

Note that $\frac{\operatorname{Var}^2(\beta_i)}{\operatorname{Var}^2(\alpha_i)} = \frac{\operatorname{Var}(\gamma_i)}{\operatorname{Var}(\alpha_i) + \operatorname{Var}(\gamma_i)} < 1$.

As discussed in Section 5.4.1, the optimal budget allocation between threshold noise and query noises within SVT with Gap is the ratio $1 : \sqrt{4k^2}$. Under this setting, we have $\operatorname{Var}(\gamma_i) = 8(1 + 4k^2)^3/\epsilon^2$. Also, we know $\operatorname{Var}(\alpha_i) = 8k^2/\epsilon^2$. Therefore, $E(\beta_i-q_i))/E(|\alpha_i-q_i|^2) = \frac{\operatorname{Var}(\beta_i)}{\operatorname{Var}(\alpha_i)} = \frac{(1 + \sqrt{4k^2})^3}{(1 + \sqrt{4k^2})^3 + k^2}$. Since $\lim_{k \to \infty} \frac{(1 + \sqrt{4k^2})^3}{(1 + \sqrt{4k^2})^3 + k^2} = \frac{4}{5}$, the improvement in accuracy approaches 20% as $k$ increases. For monotonic queries, the optimal budget allocation within SVT with Gap is $1 : \sqrt{k^2}$. Then we have $\operatorname{Var}(\gamma_i) = 8(1 + \sqrt{k^2})^3/\epsilon^2$ and therefore $\frac{\operatorname{Var}(\beta_i)}{\operatorname{Var}(\alpha_i)} = \frac{(1 + \sqrt{k^2})^3}{(1 + \sqrt{k^2})^3 + k^2}$ which is close to 50% when $k$ is large. When the algorithm uses exponential noise, the variance of the gap further reduces to $\operatorname{Var}(\gamma_i) = 4(1 + \sqrt{k^2})^3/\epsilon^2$ and therefore $\frac{\operatorname{Var}(\beta_i)}{\operatorname{Var}(\alpha_i)} = \frac{(1 + \sqrt{k^2})^3}{(1 + \sqrt{k^2})^3 + 2k^2}$ which is close to a 66% reduction of mean squared errors when $k$ is large. Our experiments in Section 5.8 confirm this improvement.

### 5.5 Improving Report Noisy Max

In this section, we present novel variations of the Noisy Max mechanism [24]. Given a list of queries with sensitivity 1, the purpose of Noisy Max is to estimate the identity (i.e., index) of the largest query. We show that, in addition to releasing this index, it is possible to release a numerical estimate of the gap between the values of the largest and second largest queries. This extra information comes at no additional cost to privacy, meaning that the original Noisy Max mechanism threw away useful information. This result can be generalized to the setting in which one wants to estimate the identities of
the top $k$ queries - we can release (for free) all of the gaps between each top $k$ query and the next best query (i.e., the gap between the best and second best queries, the gap between the second and third best queries, etc). When a user subsequently asks for a noisy answer to each of the returned queries, we show how the gap information can be used to reduce squared error by up to 66% (for counting queries).

5.5.1 Noisy Top-K with Gap

Our proposed Noisy Top-K with Gap mechanism is shown in Algorithm 18 (the function $\text{arg max}_c$ returns the top $c$ items). We can obtain the classical Noisy Max algorithm [24] from it by setting $k = 1$ and throwing away the gap information (the boxed items on Lines 6 and 7). The Noisy Top-K with Gap algorithm takes as input a sequence of $n$ queries $q_1, \ldots, q_n$, each having sensitivity 1. It adds Laplace noise to each query. It returns the indexes $j_1, \ldots, j_k$ of the $k$ queries with the largest noisy values in descending order. Furthermore, for each of these top $k$ queries $q_{j_i}$, it releases the noisy gap between the value of $q_{j_i}$ and the value of the next best query. Our key contribution in this section is the observation that these gaps can be released for free. That is, the classical Top-K algorithm, which does not release the gaps, satisfies $\varepsilon$-differential privacy. But, our improved version has exactly the same privacy cost yet is strictly better because of the extra information it can release. We emphasize that keeping the noisy gaps hidden does not decrease the privacy cost. Furthermore, this algorithm gives estimates of the pairwise gaps between any pair of the $k$ queries it selects. For example, suppose we are interested in estimating the gap between the $a^{th}$ largest and $b^{th}$ largest queries (where $a < b \leq k$). This is equal to $\sum_{i=a}^{b-1} g_i$.
because: \( \sum_{i=a}^{b-1} g_i = \sum_{i=a}^{b-1} (\tilde{q}_{ji} - \tilde{q}_{ji+1}) = \tilde{q}_{ja} - \tilde{q}_{jb} \) and hence its variance is \( \text{Var}(\tilde{q}_{ja} - \tilde{q}_{jb}) = 16k^2/\epsilon^2 \).

The original Noisy Top-K mechanism satisfies \( \epsilon \)-differential privacy. In the special case that all the \( q_i \) are counting queries then it satisfies \( \epsilon/2 \)-differential privacy [24]. We will show the same properties for Noisy Top-K with Gap. We prove the privacy property in this section and then in Section 5.5.3 we show how to use this gap information.

Local alignment. To prove the privacy of Algorithm 18, we need to create a local alignment function for each possible pair \( D \sim D' \) and output \( \omega \). Note that our mechanism uses precisely \( n \) random variables. Let \( H = (\eta_1, \eta_2, \ldots) \) where \( \eta_i \) is the noise that should be added to the \( i^{th} \) query. We view the output \( \omega = ((j_1, g_1), \ldots, (j_k, g_k)) \) as \( k \) pairs where in the \( i^{th} \) pair \( (j_i, g_i) \), the first component \( j_i \) is the index of the \( i^{th} \) largest noisy query and the second component \( g_i \) is the gap in noisy value between the \( i^{th} \) and \( (i+1)^{th} \) largest noisy queries. As in prior work [24], we will base our analysis on continuous noise so that the probability of ties among the top \( k + 1 \) noisy queries is 0. Thus each gap is positive: \( g_i > 0 \).

Let \( I_{\omega} = \{j_1, \ldots, j_k\} \) and \( I_{\omega}^c = \{1, \ldots, n\} \setminus I_{\omega} \). I.e., \( I_{\omega} \) is the index set of the \( k \) largest noisy queries selected by the algorithm and \( I_{\omega}^c \) is the index set of all unselected queries. For \( H \in \mathcal{H}_{D,\omega} \) define \( \phi_{D, D', \omega}(H) = H' = (\eta_1', \eta_2', \ldots) \) as

\[
\eta'_i = \begin{cases} 
\eta_i & i \in I_{\omega}^c \\
\eta_i + q_i - q_i' + \max_{l \in I_{\omega}^c}(q_l' + \eta_l) - \max_{l \in I_{\omega}^c}(q_l + \eta_l) & i \in I_{\omega} 
\end{cases}
\]  

(5.2)

The idea behind this local alignment is simple: we want to keep the noise of the losing queries the same (when the input is \( D \) or its neighbor \( D' \)). But, for each of the \( k \) selected queries, we want to align its noise to make sure it wins by the same amount when the input is \( D \) or its neighbor \( D' \).

Lemma 5.5. Let \( M \) be the Noisy Top-K with Gap algorithm. For all \( D \sim D' \) and \( \omega \), the functions \( \phi_{D, D', \omega} \) defined above are acyclic local alignments for \( M \). Furthermore, for every pair \( D \sim D' \), there are countably many distinct \( \phi_{D, D', \omega} \).

Proof. Given \( D \sim D' \) and \( \omega = ((j_1, g_1), \ldots, (j_k, g_k)) \), for any \( H = (\eta_1, \eta_2, \ldots) \) such that \( M(D, H) = \omega \), let \( H' = (\eta_1', \eta_2', \ldots) = \phi_{D, D', \omega}(H) \). We show that \( M(D', H') = \omega \). Since \( \phi_{D, D', \omega} \) is identity on
components \( i \in I_\omega^c \), we have \( \max_{l \in I_\omega^c} (q'_{i} + \eta'_l) = \max_{l \in I_\omega^c} (q'_l + \eta_l) \). From (5.2) we have that when \( i \in I_\omega^c \),

\[
\eta'_i = \eta_i + q_i - q'_i + \max_{l \in I_\omega^c} (q'_l + \eta_l) - \max_{l \in I_\omega^c} (q_l + \eta_l)
\]

\[\implies q'_i + \eta'_i - \max_{l \in I_\omega^c} (q'_l + \eta_l) = q_i + \eta_i - \max_{l \in I_\omega^c} (q_l + \eta_l)\]

\[\implies q'_i + \eta'_i - \max_{l \in I_\omega^c} (q'_l + \eta_l) = q_i + \eta_i - \max_{l \in I_\omega^c} (q_l + \eta_l)\]

So, for the \( k \)th selected query, \((q'_{j_k} + \eta'_{j_k}) - \max_{l \in I_\omega^c} (q'_l + \eta_l) = (q_{j_k} + \eta_{j_k}) - \max_{l \in I_\omega^c} (q_l + \eta_l) = g_k > 0 \). This means on \( D' \) the noisy query with index \( j_k \) is larger than the best of the unselected noisy queries by the same margin as it is on \( D \). Furthermore, for all \( 1 \leq i < k \), we have

\[
(q'_{j_i} + \eta'_{j_i}) - (q'_{j_{i+1}} + \eta'_{j_{i+1}})
\]

\[=(q_{j_i} + \eta_{j_i} + \max_{l \in I_\omega^c} (q'_l + \eta_l) - \max_{l \in I_\omega^c} (q_l + \eta_l))
\]

\[- (q_{j_{i+1}} + \eta_{j_{i+1}} + \max_{l \in I_\omega^c} (q'_l + \eta_l) - \max_{l \in I_\omega^c} (q_l + \eta_l))
\]

\[= (q_{j_i} + \eta_{j_i}) - (q_{j_{i+1}} + \eta_{j_{i+1}}) = g_i > 0.
\]

In other words, the query with index \( j_i \) is still the \( i \)th largest query on \( D' \) by the same margin. Therefore, \( M(D', H') = \omega \).

The local alignments are clearly acyclic (any permutation that puts \( I_\omega^c \) before \( I_\omega \) does the trick).

Also, note that \( \phi_{D', D''} \) only depends on \( \omega \) through \( I_\omega \) (the indexes of the \( k \) largest queries). There are \( n \) queries and therefore \( \binom{n}{k} = \frac{n!}{(n-k)!k!} \) distinct \( \phi_{D', D''} \). \( \square \)

Alignment cost and privacy. To establish the alignment cost, we need the following lemma.

**Lemma 5.6.** Let \((x_1, \ldots, x_m), (x'_1, \ldots, x'_m) \in \mathbb{R}^m \) be such that \( \forall i, |x_i - x'_i| \leq 1 \). Then \( \max_i (|x_i - \max_j (x'_j)|) \leq 1 \).

**Proof.** Let \( s \) be an index that maximizes \( x_s \) and let \( t \) be an index that maximizes \( x'_s \). Without loss of generality, assume \( x_s \geq x'_s \). Then \( x_s \geq x'_s \geq x'_t \geq x_s - 1 \). Hence \( |x_s - x'_s| = x_s - x'_s \leq x_s - (x_s - 1) = 1 \). \( \square \)

**Theorem 5.7.** The Noisy Top-K with Gap mechanism satisfies \( \epsilon \)-differential privacy. If all of the queries are counting queries, then it satisfies \( \epsilon/2 \)-differential privacy.
Proof. First we bound the cost of the alignment function defined in (5.2). Recall that the $\eta_i$’s are independent Lap$(2k/\epsilon)$ random variables. By Definition 5.4
\[
\text{cost}(\phi_{D,D',\omega}) = \sum_{i=1}^{\infty} |\eta'_i - \eta_i| \frac{\epsilon}{2k} = \frac{\epsilon}{2k} \sum_{i \in I_\omega} |q_i - q'_i + \max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)|.
\]
By the global sensitivity assumption we have $|q_i - q'_i| \leq 1$. Apply Lemma 5.6 to the vectors $(q_l + \eta_l)_{l \in I_\omega}$ and $(q'_l + \eta_l)_{l \in I_\omega}$, we have $|\max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)| \leq 1$. Therefore,
\[
|q_i - q'_i + \max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)| \leq |q_i - q'_i| + |\max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)| \leq 1 + 1 = 2.
\]
Furthermore, if $q$ is monotonic, then

- either $\forall i : q_i \leq q'_i$ in which case $q_i - q'_i \in [-1, 0]$ and $\max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l) \in [0, 1]$,

- or $\forall i : q_i \geq q'_i$ in which case $q_i - q'_i \in [0, 1]$ and $\max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l) \in [-1, 0]$.

In both cases we have $q_i - q'_i + \max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l) \in [-1, 1]$ so $|q_i - q'_i + \max_{l \in I_\omega} (q'_i + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)| \leq 1$. Therefore,
\[
\text{cost}(\phi_{D,D',\omega}) \leq \frac{\epsilon}{2k} \sum_{i \in I_\omega} 2 \quad \text{(or} \quad \frac{\epsilon}{2k} \sum_{i \in I_\omega} 1 \text{if} \; q \; \text{is monotonic)}
\]
\[
= \frac{\epsilon}{2k} \cdot 2 |I_\omega| \quad \text{(or} \quad \frac{\epsilon}{2k} \cdot |I_\omega| \text{if} \; q \; \text{is monotonic)}
\]
\[
= \epsilon \quad \text{(or} \quad \epsilon/2 \text{if} \; q \; \text{is monotonic)}.
\]

Conditions 1 through 3 of Lemma 5.1 are trivial to check, 4 and 5 follow from Lemma 5.5 and the above bound on cost. Therefore, Theorem 5.7 follows from Lemma 5.1. \qed
5.5.2 Noisy Top-K with Exponential Noise

The original noisy max algorithm also works with one-sided exponential noise [24] with smaller variance than the Laplace noise. In this subsection, we show that this result extends to the Noisy Top-K with Gap algorithm by simply changing Line 3 of Algorithm 18 to $\eta_i \leftarrow \text{Exp}(2k/\epsilon)$ and privacy is maintained while the variance of the gap decreases. However, the proof relies on a different local alignment.

**Local alignment.** The alignment used in Section 5.5.1 will not work here because it might set our noise random variables to negative numbers. Thus we need a new alignment. As before, let $H = (\eta_1, \eta_2, \ldots)$ where $\eta_i$ is the noise that should be added to the $i^{th}$ query. We view the output $\omega = ((j_1, g_1), \ldots, (j_k, g_k))$ as $k$ pairs where in the $i^{th}$ pair $(j_i, g_i)$, the first component $j_i$ is the index of $i^{th}$ largest noisy query and the second component $g_i > 0$ is the gap in noisy value between the $i^{th}$ and $(i + 1)^{th}$ largest noisy queries.

Let $I_\omega = \{j_1, \ldots, j_k\}$ and $I_\omega^c = \{1, \ldots, n\} \setminus I_\omega$. I.e., $I_\omega$ is the index set of the $k$ largest noisy queries selected by the algorithm and $I_\omega^c$ is the index set of all unselected queries. For $H \in \mathcal{H}_{D,\omega}$ we will use $\phi_{D,D',\omega}(H) = H' = (\eta'_1, \eta'_2, \ldots)$ to refer to the aligned noise. In order to define the alignment, we need the following quantities:

$$s = \arg\max_{l \in I_\omega^c} (q_l + \eta_l), \quad t = \arg\max_{l \in I_\omega^c} (q'_l + \eta_l)$$

$$i^* = \arg\min_{i \in I_\omega} \{q_i - q'_i + \max_{l \in I_\omega} (q'_l + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)\}$$

$$= \arg\min_{i \in I_\omega} \{q_i - q'_i\} \quad \text{(the other terms have no i)}$$

$$\delta^* = \min_{i \in I_\omega} \{q_i - q'_i + \max_{l \in I_\omega} (q'_l + \eta_l) - \max_{l \in I_\omega} (q_l + \eta_l)\}$$

$$= q_{i^*} - q'_{{i^*}} + (q'_{{i^*}} + \eta_{{i^*}}) - (q_{{s}} + \eta_{{s}})$$

Note that $i^* \in I_\omega$ and $s, t \in I_\omega^c$. We define the alignment according to the value of $\delta^*$. When $\delta^* \geq 0$, we
use the same alignment as in the Laplace version of the algorithm:

\[
\eta_i' = \begin{cases} 
\eta_i & i \in I_\omega^c \\
\eta_i + q_i - q_i' + (q_i' + \eta_i) - (q_s + \eta_s) & i \in I_\omega 
\end{cases} 
\quad (5.3)
\]

When \(\delta_s < 0\) that alignment could result in a negative \(\eta_i'\) for some \(i \in I_\omega\). So instead, we take that alignment and further add the positive quantity \(-\delta_s\) in several places so that overall we are adding nonnegative numbers to each \(\eta_i\) to get \(\eta_i'\) (this ensures that \(\eta_i'\) is nonnegative for each \(i\)). Thus, when \(\delta_s < 0\), define

\[
\eta_i' = \begin{cases} 
\eta_i & i \in I_\omega^c \setminus \{t\} \\
\eta_i - \delta_s & i = t \\
\eta_i + q_i - q_i' + (q_i' + \eta_i) - (q_s + \eta_s) - \delta_s & i \in I_\omega 
\end{cases} 
\quad (5.4)
\]

**Lemma 5.8.** Let \(M\) be the Noisy Top-K with Gap algorithm that uses exponential noise. For all \(D \sim D'\) and \(\omega\), the functions \(\phi_{D,D',\omega}\) defined above are acyclic local alignments for \(M\). Furthermore, for every pair \(D \sim D'\), there are countably many distinct \(\phi_{D,D',\omega}\).

**Proof.** First we show that \(\forall i, \eta_i' \geq \eta_i\). Recall that \(\delta_s = \min_{i \in I_\omega} \{q_i - q_i' + (q_i' + \eta_i) - (q_s + \eta_s)\}\). When \(\delta_s \geq 0\), we have \(\eta_i' - \eta_i = q_i - q_i' + (q_i' + \eta_i) - (q_s + \eta_s) \geq \delta_s \geq 0\) for all \(i \in I_\omega\). When \(\delta_s < 0\), we have \(\eta_i' - \eta_i = -\delta_s > 0\) and \(\eta_i' - \eta_i = (q_i - q_i') - (q_i - q_i') \geq 0\) for \(i \in I_\omega\). Therefore, all \(\eta_i'\) are non-negative.

The proof that (5.3) is an alignment when \(\delta_s \geq 0\) is the same as in the Laplace noise case. To show that (5.4) is an alignment when \(\delta_s < 0\), first note that since \(t = \arg\max_{i \in I_\omega} (q_i' + \eta_i)\) and \(-\delta_s > 0\), we have \(t = \arg\max_{i \in I_\omega} (q_i' + \eta_i)\). Then from (5.4), we have that when \(i \in I_\omega\),

\[
\eta_i' = \eta_i + q_i - q_i' + (q_i' + \eta_i) - (q_s + \eta_s) - \delta_s \\
\implies q_i' + \eta_i' - (q_i' + (\eta_i - \delta_s)) = q_i + \eta_i - (q_s + \eta_s)
\]
\[ q'_i + \eta'_i - (q'_i + \eta'_i) = q_i + \eta_i - (q_s + \eta_s) \]
\[ q'_i + \eta'_i - \max_{l \in \mathcal{I}_a} (q'_i + \eta'_i) = q_i + \eta_i - \max_{l \in \mathcal{I}_a} (q_l + \eta_l) \]

Thus by a similar argument in Lemma 5.5, all relative orders among the \( k \) largest noisy queries and their associated gaps are preserved. The facts that \( \phi\mathcal{D,\mathcal{D}',\omega} \) is acyclic and there are finitely many \( \phi\mathcal{D,\mathcal{D}',\omega} \) are clear. \( \square \)

**Alignment cost and privacy.** Recall from Table 5-1 that if \( f(x) \) is the density of \( \text{Exp}(\beta) \), then for \( x, y \geq 0 \),
\[ \ln \frac{f(x)}{f(y)} = \frac{y-x}{\beta} \leq \frac{|y-x|}{\beta}. \]
When \( \delta_* \geq 0 \), the alignment cost computation is the same as with the Laplace version of the algorithm. When \( \delta_* < 0 \), we have
\[ \text{cost}(\phi\mathcal{D,\mathcal{D}',\omega}) = \sum_{i=1}^{\infty} |\eta'_i - \eta_i| \cdot \frac{e}{2k} \]
\[ = \frac{e}{2k} |\delta_*| + \frac{e}{2k} \sum_{i \in \mathcal{I}_a} |q_i - q'_i - q_i + q'_i| \]
\[ = \frac{e}{2k} |\delta_*| + \frac{e}{2k} \sum_{i \in \mathcal{I}_a \setminus \{i_*\}} |q_i - q'_i - q_i + q'_i|. \]

and note that there are \( k - 1 \) terms in the right-most summation. It is clear that \( |q_i - q'_i - q_i + q'_i| \leq 2 \) (or 1 if \( q \) is monotone). Also, it is shown in the proof of Theorem 5.7 that
\[ |\delta_*| = |q_i - q'_i + \max_{l \in \mathcal{I}_a} (q'_i + \eta_i) - \max_{l \in \mathcal{I}_a} (q_l + \eta_l)| \leq 2 \] (or 1 if \( q \) is monotone). Therefore,
\[ \text{cost}(\phi\mathcal{D,\mathcal{D}',\omega}) = \frac{e}{2k} |\delta_*| + \frac{e}{2k} \sum_{i \in \mathcal{I}_a \setminus \{i_*\}} |q_i - q'_i - q_i + q'_i| \]
(note that there are \( 1 + (k - 1) = k \) terms above)
\[ \leq \frac{e}{2k} \cdot 2 \cdot k \quad (\text{or } \frac{e}{2k} \cdot 1 \cdot k \text{ if } q \text{ is monotonic}) \]
\[ = e \quad (\text{or } e/2 \text{ if } q \text{ is monotonic}). \]

Thus, Algorithm 18 with \( \text{Exp}(2k/e) \) noise on Line 3 instead of \( \text{Lap}(2k/e) \) noise, satisfies \( \epsilon \)-differential privacy. If all of the queries are counting queries, then it satisfies \( \epsilon/2 \)-differential privacy.
5.5.3 Utilizing Gap Information

Let us consider one scenario that takes advantage of the gap information. Suppose a data analyst is interested in the identities and values of the top \( k \) queries. A typical approach would be to split the privacy budget \( \epsilon \) in half – use \( \frac{\epsilon}{2} \) of the budget to identify the top \( k \) queries using Noisy Top-K with Gap. The remaining \( \frac{\epsilon}{2} \) budget is evenly divided between the selected queries and is used to obtain noisy measurements (i.e. add \( \text{Lap}(2k/\epsilon) \) noise to each query answer). These measurements will have variance \( \sigma^2 = \frac{8k^2}{\epsilon^2} \). In this section we show how to use the gap information from Noisy Top-K with Gap and postprocessing to improve the accuracy of these measurements.

**Problem statement.** Let \( q_1, \ldots, q_k \) be the true answers of the top \( k \) queries that are selected by Algorithm 18. Let \( \alpha_1, \ldots, \alpha_k \) be their noisy measurements. Let \( g_1, \ldots, g_{k-1} \) be the noisy gaps between \( q_1, \ldots, q_k \) that are obtained from Algorithm 18 for free. Then \( \alpha_i = q_i + \xi_i \) where each \( \xi_i \) is a \( \text{Lap}(2k/\epsilon) \) random variable and \( g_i = q_i + \eta_i - q_{i+1} - \eta_{i+1} \) where each \( \eta_i \) is a \( \text{Lap}(4k/\epsilon) \) random variable, or a \( \text{Lap}(2k/\epsilon) \) random variable if the query list is monotonic (recall the mechanism was run with a privacy budget of \( \frac{\epsilon}{2} \)). Our goal is then to find the best linear unbiased estimate (BLUE) \([93]\) \( \beta_i \) of \( q_i \) in terms of the measurements \( \alpha_i \) and gap information \( g_i \).

**Theorem 5.9.** With above notation let \( \mathbf{q} = [q_1, \ldots, q_k]^T \), \( \mathbf{\alpha} = [\alpha_1, \ldots, \alpha_k]^T \) and \( \mathbf{g} = [g_1, \ldots, g_{k-1}]^T \). Suppose the ratio \( \text{Var}(\xi_i) : \text{Var}(\eta_i) \) is equal to \( 1 : \lambda \). Then the BLUE of \( \mathbf{q} \) is \( \mathbf{\beta} = \frac{1}{(1+\lambda)k} (X\mathbf{\alpha} + Y\mathbf{g}) \) where

\[
X = \begin{pmatrix}
1 + \lambda k & 1 & \cdots & 1 \\
1 & 1 + \lambda k & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \cdots & 1 + \lambda k
\end{pmatrix}_{k \times k}
\]

\[
Y = \begin{pmatrix}
k - 1 & k - 2 & \cdots & 1 \\
k - 1 & k - 2 & \cdots & 1 \\
k - 1 & k - 2 & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
k - 1 & k - 2 & \cdots & 1 \\
\end{pmatrix}_{k \times (k-1)}
\]
For proof, see Section 5.10. Even though this is a matrix multiplication, it is easy to see that it translates into the following algorithm that is linear in $k$:

1. Compute $\alpha = \sum_{i=1}^{k} \alpha_i$ and $p = \sum_{i=1}^{k-1} (k-i) g_i$.
2. Set $p_0 = 0$. For $i = 1, \ldots, k-1$ compute the prefix sum $p_i = \sum_{j=1}^{i} g_j = p_{i-1} + g_i$.
3. For $i = 1, \ldots, k$, set $\beta_i = (\alpha + \lambda k \alpha_i + p - k p_{i-1}) / (1 + \lambda) k$.

Now, each $\beta_i$ is an estimate of the value of $q_i$. How does it compare to the direct measurement $\alpha_i$ (which has variance $\sigma^2 = 8k^2/\epsilon^2$)? The following result compares the expected error of $\beta_i$ (which used the direct measurements and the gap information) with the expected error of using only the direct measurements (i.e., $\alpha_i$ only).

**Corollary 5.9.1.** For all $i = 1, \ldots, k$, we have

$$\frac{E(|\beta_i - q_i|^2)}{E(|\alpha_i - q_i|^2)} = \frac{1 + \lambda k}{k + \lambda k} = \frac{\text{Var}(\xi_i) + k \text{Var}(\eta_i)}{k (\text{Var}(\xi_i) + \text{Var}(\eta_i))}.$$

For proof, see the Section 5.10. In the case of counting queries, we have $\text{Var}(\xi_i) = \text{Var}(\eta_i) = 8k^2/\epsilon^2$ and thus $\lambda = 1$. The error reduction rate is $\frac{k-1}{2k}$ which is close to 50% when $k$ is large. If we use exponential noise instead, i.e., replace $\eta_i \leftarrow \text{Lap}(2k/\epsilon)$ with $\eta_i \leftarrow \text{Exp}(2k/\epsilon)$ at Line 3 of Algorithm 18, then $\text{Var}(\eta_i) = 4k^2/\epsilon^2 = \text{Var}(\xi_i)/2$ and thus $\lambda = 1/2$. In this case, the error reduction rate is $\frac{2k-2}{3k}$ which is close to 66% when $k$ is large. Our experiments in Section 5.8 confirm these theoretical results.

### 5.6 SVT/Noisy Max Hybrids with Gap

In this section, we present two hybrids of SVT with Gap and Noisy Top-K with Gap. Recall that SVT with Gap is an online algorithm that returns the identities and noisy gaps (with respect to the threshold) of the first $k$ noisy queries it sees that are larger than the noisy threshold. Its benefits are:

(i) Privacy budget is saved if fewer than $k$ queries are returned.

(ii) The queries that are returned come with estimates of their noisy answers (obtained by adding the public threshold to the noisy gap).
while the drawbacks are that the returned queries are likely not to resemble the $k$ largest queries (queries that come afterwards are ignored, no matter how large their values are).

Meanwhile, Noisy Top-K with Gap returns the identities and gaps (with respect to the runner-up query) of the top $k$ noisy queries. Its benefits are:

(i) The queries returned are approximately the top $k$.

(ii) The gap tells us how large the queries are compared to the best non-selected noisy query.

The drawbacks are:

(i) $k$ queries are always returned, even if their values are small.

(ii) Only gap information is returned (not estimates of the query answers).

For users who are interested in identifying the top $k$ queries that are likely to be over a threshold, we present two hybrid algorithms that try to combine the benefits of both algorithms while minimizing the drawbacks. Both algorithms take as input a number $k$, a list of answers to queries having sensitivity 1, and a public threshold $T$. They both return a subset of the top $k$ noisy queries that are larger than the noisy threshold $T$, hence the privacy cost is dynamic and is smaller if fewer than $k$ queries are returned. The difference is in the gap information.

The first hybrid (Algorithm 19) is a variant of Noisy Top-K with Gap. It adds the public threshold $T$ to the list of queries (it becomes Query 0), adds the same noise to them (Lines 2 and 4). In line 6, it takes the top $k$ noisy queries (sorted in decreasing order) and their gaps with the next best query. It filters out any that are smaller than the noisy Query 0. For the queries that didn’t get removed, it returns their identities (recall the threshold is Query 0) and their gap with the next best query. If the last returned item is Query 0, this means that the gap information tells us how much larger the other returned queries are compared to the noisy threshold Query 0, and this allows us to get numerical estimates for those query answers by adding in the public threshold.

Alignment and privacy cost for Algorithm 19. By replacing the index sets $I_\omega$ in Equations (5.3) and (5.4) with $I_\omega = \{j_1, \ldots, j_t\}$, the same formula can be used as the alignment function for Algorithm 19. Note that since $|I_\omega| = t \leq k$, the privacy cost is $(t/k)\epsilon$. 
Algorithm 19: Hybrid Noisy Top-K with Gap

input : $q$: a list of $n$ queries of global sensitivity 1
$D$: database, $\epsilon$: privacy budget
$T$: public threshold, $k$: # of indexes

1 function NoisyTopK $(q, D, T, k, \epsilon)$:
2 $\eta_0 \leftarrow \exp(2k/\epsilon)$; $\widetilde{q}_0 \leftarrow T + \eta_0$
3 foreach $i \in \{1, \ldots, n\}$ do
4 $\eta_i \leftarrow \exp(2k/\epsilon)$; $\widetilde{q}_i \leftarrow q_i(D) + \eta_i$
5 $(j_1, \ldots, j_{k+1}) \leftarrow \arg \max_{k+1}(\widetilde{q}_0, \widetilde{q}_1, \ldots, \widetilde{q}_n)$
6 foreach $i \in \{1, \ldots, k\}$ do
7 $g_i \leftarrow \widetilde{q}_{j_i} - \widetilde{q}_{j_{i+1}}$; $t \leftarrow i$
8 if $j_i = 0$ then
9 break
10 return $((j_1, g_1), \ldots, (j_t, g_t))$

Lemma 5.10. If Algorithm 19 is run with privacy budget $\epsilon$ and returns $t$ queries (and their associated gaps), then the actual privacy cost is $(t/k)\epsilon$.

The second hybrid (Algorithm 20) is essentially SVT with Gap applied to the list of queries that is sorted in descending order by their noisy answers. We note that it adds more noise to each query than Algorithm 19 but always returns the noisy gap between the noisy query answer and the noisy threshold, just like SVT with Gap.

Algorithm 20: Hybrid Sparse Vector with Gap

input : same as Algorithm 19

1 function GapSparse $(q, D, T, k, \epsilon)$:
2 $\epsilon_0 \leftarrow \theta \epsilon$; $\epsilon_1 \leftarrow (1 - \theta)\epsilon/k$;
3 $\eta \leftarrow \exp(1/\epsilon_0)$; $\widetilde{T} \leftarrow T + \eta - 1/\epsilon_0$
4 foreach $i \in \{1, \ldots, n\}$ do
5 $\eta_i \leftarrow \exp(2/\epsilon_1)$; $\widetilde{q}_i \leftarrow q_i(D) + \eta_i - 2/\epsilon_1$
6 $(j_1, \ldots, j_k) \leftarrow \arg \max_k(\widetilde{q}_1, \ldots, \widetilde{q}_n)$
7 $t \leftarrow 0$
8 foreach $i \in \{1, \ldots, k\}$ do
9 if $\widetilde{q}_{j_i} \geq \widetilde{T}$ then
10 $g_i \leftarrow \widetilde{q}_{j_i} - \widetilde{T}$; $t \leftarrow i$
11 else
12 break
13 return $((j_1, g_1), \ldots, (j_t, g_t))$ // $\emptyset$ if $t = 0$
Alignment and privacy cost for Algorithm 20. The alignment for Algorithm 20 is the same as the one for SVT with Gap and is hence omitted here. Note that the privacy cost is $\epsilon_0 + t\epsilon_1 = (\theta + (t/k)(1 - \theta))*\epsilon$ where $t$ is the number of queries returned. As discussed in Section 5.4.1, the optimal $\theta$ is $1/(1 + \sqrt{4k^2})$.

**Lemma 5.11.** If Algorithm 20 is run with privacy budget $\epsilon$ and returns $t$ queries (and their associated gaps), then the actual privacy cost is $(\theta + (t/k)(1 - \theta))*\epsilon$.

Benefits of the Hybrid Algorithms. Compared with Noisy Top-K with Gap, the hybrid algorithms have these advantages:

(i) saving privacy budget: The actual privacy budget consumption for the hybrid algorithms is dynamic – it depends on the number of queries returned. Thus if the threshold $T$ is set high, the hybrid algorithms will likely return fewer than $k$ queries and consume less privacy budget;

(ii) providing query estimates: Algorithm 20 always returns the noisy gap with the threshold (hence, by adding in the public threshold value, this gives an estimate of the query answer). Meanwhile, Algorithm 19 only returns the noisy gap with the threshold if the last query returned is the noisy threshold Query 0 (otherwise it functions like Noisy Top-K with Gap and returns the gaps with the runner up query).

Compared with SVT with Gap, the hybrid algorithms are trying to select the overall top $k$ queries that are above the threshold, whereas SVT with Gap tries pick the first $k$ queries it sees that are above the threshold. So the queries returned by the hybrid algorithms are expected to have much higher values. There is an important distinction though: SVT with Gap is an online algorithm that can process queries as they arrive, whereas the hybrid algorithms require all queries to be known beforehand.

The first hybrid (Algorithm 19) is more likely to provide accurate identity information than the second hybrid (Algorithm 20). That is, the queries it returns are more likely to be the actual queries whose true values are largest (because the first algorithm adds less noise to the query answers). However, as mentioned before Algorithm 20 always provide estimates of query answers, whereas Algorithm 19 only provide such estimates if the last query returned is the noisy threshold Query 0. Therefore, if it is more desirable to always have query answer estimates then one should use Algorithm 20. Otherwise Algorithm 19 is a good default choice.
5.7 Improving the Exponential Mechanism

The Exponential Mechanism [25] was designed to answer non-numeric queries in a differentially private way. In this setting, $\mathcal{D}$ is the set of possible input databases and $\mathcal{R} = \{\omega_1, \omega_2, \ldots, \omega_n\}$ is a set of possible outcomes. There is a utility function $\mu : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$ where $\mu(D, \omega_i)$ gives us the utility of outputting $\omega_i$ when the true input database is $D$. The exponential mechanism randomly selects an output $\omega_i$ with probabilities that are defined by the following theorem:

**Theorem 5.12 (The Exponential Mechanism [25]).** Given $\epsilon > 0$ and a utility function $\mu : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$, the mechanism $M(D, \mu, \epsilon)$ that outputs $\omega_i \in \mathcal{R}$ with probability proportional to $\exp\left(\frac{\epsilon \mu(D, \omega_i)}{2\Delta_{\mu}}\right)$ satisfies $\epsilon$-differential privacy where $\Delta_{\mu}$, the sensitivity of $\mu$, is defined as

$$\Delta_{\mu} = \max_{D \sim D'} \max_{\omega_i \in \mathcal{R}} |\mu(D, \omega_i) - \mu(D', \omega_i)|.$$

We show that the Exponential Mechanism can also output (for free) a type of gap information in addition to the selected index. This gap provides noisy information about the difference between the utility scores of the selected output and non-selected outputs. What is surprising about this result is that we can treat the exponential mechanism as a black box (i.e., it doesn’t matter how the sampling is implemented). In contrast, the internal state of the Noisy Max algorithm was needed (i.e., the gap was computed from the noisy query answers). The details are shown in Algorithm 21, which makes use of the Logistic($\theta$) distribution having pdf $f(x; \theta) = \frac{e^{-(x-\theta)}}{1+e^{-(x-\theta)}}$.

**Algorithm 21: Exponential Mechanism w. Gap**

*input*: $\mu$: utility function with sensitivity $\Delta_{\mu}$

$D$: database, $\epsilon$: privacy budget

1. **function** GapExpMech ($D$, $\mu$, $\epsilon$):
   2. $\omega_s \leftarrow \text{ExpMech}(D, \mu, \epsilon)$ // Selected query
   3. $\theta \leftarrow \frac{\epsilon \mu(D, \omega_s)}{2\Delta_{\mu}} - \ln \sum_{j \neq s} \exp\left(\frac{\epsilon \mu(D, \omega_j)}{2\Delta_{\mu}}\right)$
   4. **while** true **do**
      5. $g_s \leftarrow \text{Logistic}(\theta)$ // Location=$\theta$, scale=1
      6. **if** $g_s > 0$ **then**
         7. break
   8. **return** $\omega_s, g_s$
**Theorem 5.13.** Algorithm 21 satisfies $\epsilon$-differential privacy and the expected value of $g_s$ is \((1 + e^{-\theta}) \ln(1 + e^\theta)\) where $s$ is the index of the query returned by the exponential mechanism and $\theta = \frac{\epsilon \mu_{(D, \omega_s)}}{2 \Delta \mu} - \ln \sum_{j \neq s} \exp\left(\frac{\epsilon \mu_{(D, \omega_j)}}{2 \Delta \mu}\right)$ is the location parameter of the sampling distribution.

**Utilizing the Gap Information.** From Theorem 5.13 and Algorithm 21, we see that $\theta$ is a kind of gap (scaled by $\epsilon/2\Delta \mu$) between the selected query $\omega_s$ and a softmax of the remaining items. While $\theta$ can be numerically estimated from $g_s$, one can also use $g_s$ for the following purpose.

The exponential mechanism is randomized, so an important question is whether it returned a query that has the highest utility. We can use the noisy gap information $g_s$ from Algorithm 21 to answer this question in a hypothesis testing framework. Specifically, let $H_0$ be the null hypothesis that the returned query $\omega_s$ does not have the highest utility score. Then $g_s$ can tell us how unlikely this null hypothesis is – the quantity $P(g_s \geq \gamma \mid H_0)$ is the significance level (also known as a p-value), and small values indicate the null hypothesis is unlikely. Its computation is given in Theorem 5.14.

**Theorem 5.14.** $P(g_s \geq \gamma \mid H_0) \leq \frac{2}{1 + e^\gamma}$.

We note that if we want a significance level of $\alpha = 0.05$ (i.e., there is less than a 5% chance that a non-optimal query could have produced a large noisy gap) then we need $g_s \geq \ln\left(\frac{2}{\alpha} - 1\right) \approx 3.66$.

### 5.8 Experiments

We evaluate the algorithms proposed in this paper using the two real datasets from [11]: BMP-POS, Kosarak and a synthetic dataset T40I10D100K created by the generator from the IBM Almaden Quest research group. These datasets are collections of transactions (each transaction is a set of items). In our experiments, the queries correspond to the counts of each item (i.e. how many transactions contained item #23?) The statistics of the datasets are listed below in Table 5-3.
Table 5-3: Statistics of Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of Records</th>
<th># of Unique Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS-POS</td>
<td>515,597</td>
<td>1,657</td>
</tr>
<tr>
<td>Kosarak</td>
<td>990,002</td>
<td>41,270</td>
</tr>
<tr>
<td>T40I10D100K</td>
<td>100,000</td>
<td>942</td>
</tr>
</tbody>
</table>

5.8.1 Improving Query Estimates with Gap Information

The first set of experiments is to measure how gap information can help improve estimates in selected queries. We use the setup of Sections 5.5.3 and 5.4.3. That is, a data analyst splits the privacy budget $\epsilon$ in half. She uses the first half to select $k$ queries using Noisy Top-K with Gap or SVT with Gap (or Adaptive SVT with Gap) and then uses the second half of the privacy budget to obtain independent noisy measurements of each selected query.

If one were unaware that gap information came for free, one would just use those noisy measurements as estimates for the query answers. The error of this approach is the gap-free baseline. However, since the gap information does come for free, we can use the postprocessing described in Sections 5.5.3 and 5.4.3 to improve accuracy (we call this latter approach SVT with Gap with Measures and Noisy Top-K with Gap with Measures).

![Figure 5-2](image)

(a) SVT with Gap with Measures, BMS-POS.  
(b) Noisy Top-K with Gap with Measures, BMS-POS.

Figure 5-2: Percent reduction of Mean Squared Error on monotonic queries, for different $k$, for SVT with Gap and Noisy Top-K with Gap when half the privacy budget is used for query selection and the other half is used for measurement of their answers. Privacy budget $\epsilon = 0.7$.

We first evaluate the percentage reduction of mean squared error (MSE) of the postprocessing
Figure 5-3: Percent reduction of Mean Squared Error on monotonic queries, for different $\epsilon$, for SVT with Gap and Noisy Top-K with Gap when half the privacy budget is used for query selection and the other half is used for measurement of their answers. The value of $k$ is set to 10.

approach compared to the gap-free baseline and compare this improvement to our theoretical analysis. As discussed in Section 5.4.3, we set the budget allocation ratio within the SVT with Gap algorithm (i.e., the budget allocation between the threshold and queries) to be $1 : k^{\frac{1}{2}}$ for monotonic queries and $1 : (2k)^{\frac{3}{2}}$ otherwise – such a ratio is recommended in [11] for the original SVT. The threshold used for SVT with Gap is randomly picked from the top $2k$ to top $8k$ in each dataset for each run. All numbers plotted are averaged over 10,000 runs. Due to space constraints, we only show experiments for counting queries (which are monotonic).

Our theoretical analysis in Sections 5.4.3 and 5.5.3 suggested that in the case of monotonic queries, the error reduction rate can reach up to 50% when Laplace noise is used, and 66% when exponential or geometric noise is used, as $k$ increases. This is confirmed in Figures 5-2a, for SVT with Gap and Figures 5-2b, for our Top-K algorithm using the BMS-POS dataset (results for the other datasets are nearly identical). These figures plot the theoretical and empirical percent reduction of MSE as a function of $k$ and show the power of the free gap information.

We also generated corresponding plots where $k$ is held fixed and the total privacy budget $\epsilon$ is varied. We only present the result for the kosarak dataset as results for the other datasets are nearly identical. For SVT with Gap, Figures 5-3a confirms that this improvement is stable for different $\epsilon$ values. For our Top-K algorithm, Figures 5-3b confirms that this improvement is also stable for different values of $\epsilon$.

---

6Selecting thresholds for SVT in experiments is difficult, but we feel this may be fairer than averaging the answer to the top $k^{th}$ and $k+1^{th}$ queries as was done in prior work [11].
5.8.2 Benefits of Adaptivity

In this section we present an evaluation of the budget-saving properties of our novel Adaptive SVT with Gap algorithm to show that it can answer more queries than SVT and SVT with Gap at the same privacy cost (or, conversely, answer the same number of queries but with leftover budget that can be used for other purposes). First note that SVT and SVT with Gap both answer exactly the same amount of queries, so we only need to compare Adaptive SVT with Gap to the original SVT [24, 11]. In both algorithms, the budget allocation between the threshold noise and query noise is set according to the ratio 1 : $k^{\frac{2}{3}}$ (i.e., the hyperparameter $\theta$ in Adaptive SVT with Gap is set to $1/(1 + k^{\frac{2}{3}})$), following the discussion in Section 5.4.1. The threshold is randomly picked from the top $2k$ to top $8k$ in each dataset and all reported numbers are averaged over 10,000 runs.
Number of queries answered. We first compare the number of queries answered by each algorithm as the parameter $k$ is varied from 2 to 24 with a privacy budget of $\epsilon = 0.7$ (results for other settings of the total privacy budget are similar). The results are shown in Figure 5-4a, 5-4b, and 5-4c. In each of these bar graphs, the left (blue) bar is the number of answers returned by SVT and the right bar is the number of answers returned by Adaptive SVT with Gap. This right bar is broken down into two components: the number of queries returned from the top “if” branch (corresponding to queries that were significantly larger than the threshold even after a lot of noise was added) and the number of queries returned from the middle “if” branch. Queries returned from the top branch of Adaptive SVT with Gap have less privacy cost than those returned by SVT. Queries returned from the middle branch of Adaptive SVT with Gap have the same privacy cost as in SVT. We see that most queries are answered in the top branch of Adaptive SVT with Gap, meaning that the above-threshold queries were generally large (much larger than the threshold). Since Adaptive SVT with Gap uses more noise in the top branch, it uses less privacy budget to answer those queries and uses the remaining budget to provide additional answers (up to an average of 20 more answers when $k$ was set to 24).

Precision and F-Measure. Although the adaptive algorithm can answer more above-threshold queries than the original, one can still ask the question of whether the returned queries really are above the threshold. Thus we can look at the precision of the returned results (the fraction of returned queries that are actually above the threshold) and the widely used F-Measure (the harmonic mean of precision and recall). One would expect that the precision of Adaptive SVT with Gap should be less than that of SVT, because the adaptive version can use more noise when processing queries. In Figures 5-5a, 5-5b, and 5-5c we compare the precision and F-Measure of the two algorithms. Generally we see very little difference in precision. On the other hand, since Adaptive SVT with Gap answers more queries while maintaining high precision, the recall of Adaptive SVT with Gap would be much larger than SVT, thus leading to the F-Measure being roughly 1.5 times that of SVT.

Remaining Privacy Budget. If a query is large, Adaptive SVT with Gap may only need to use a small part of the privacy budget to determine that the query is likely above the noisy threshold. That is, it may produce an output in its top branch, where a lot of noise (hence less privacy budget) is used. If we stop Adaptive SVT with Gap after $k$ returned queries, it may still have some privacy budget left over (in
Figure 5-6: Remaining privacy budget when Adaptive SVT with Gap is stopped after answering $k$ queries using different datasets. Privacy budget $\epsilon = 0.7$.

Contrast to standard versions of Sparse Vector, which use up all of their privacy budget). This remaining privacy budget can then be used for other data analysis tasks. For all three datasets, Figure 5-6 shows the percentage of privacy budget that is left over when Adaptive SVT with Gap is run with parameter $k$ and stopped after $k$ queries are returned. We see that roughly 40% of the privacy budget is left over, confirming that Adaptive SVT with Gap is able to save a significant amount of privacy budget.

5.8.3 Benefits of the Hybrid Algorithms

We next evaluate whether our hybrid algorithms combine the best properties of SVT (saving budget if few queries are over the threshold) and Noisy Top-K (selecting queries with higher values than SVT).

To evaluate the budget-saving properties, we set the threshold $T$ to be the 12th largest query and let $k$ vary from 2 to 24. This creates settings where fewer than $k$ queries may be returned (i.e., when $k > 12$). The remaining privacy budget for different $k$ are shown in Figure 5-7. When $k > 12$, SVT and the hybrid algorithms use less privacy budget because they return fewer than $k$ queries. However, Noisy Top-K uses the full budget because it returns $k$ queries, even when we don’t want the ones below the threshold. Hybrid Noisy Top-K saves more privacy budget than Hybrid SVT because Hybrid SVT spends a fixed amount of budget $\theta \epsilon$ on the threshold whereas Hybrid Noisy Top-K treats the threshold as a query and only spends $\epsilon/k$ on it. SVT behaves similarly to Hybrid SVT in terms of budget consumption.

Next, we compare how well the algorithms return queries whose answers are large. Using the same
settings as before, we show how the average of the answers to the returned queries (as $k$ varies) in Figure 5-8.

Since the threshold is set at the value of the 12\textsuperscript{th} largest query, when $k \leq 12$, the algorithms tend to return $k$ queries. Here Noisy Top-K and the hybrid algorithms return much better queries than SVT. However, when $k > 12$, we are only interested in the queries that are larger than the threshold. Noisy Top-K has no ability to filter out the queries below the threshold and so the average query quality decreases. Meanwhile, SVT and our hybrid algorithms filter out the queries that are likely to be below the threshold, resulting in higher average quality. Thus we see that the hybrid algorithms indeed inherit the best properties of SVT and Noisy Top-K.
5.8.4 p-Values from Exponential Mechanism with Gap

Algorithm 21 returns a selected query $\omega_s$ and a gap estimate $g_s$ that we can use for hypothesis testing. Let $H_0$ be the null hypothesis that $\omega_s$ is not the query with the highest utility. Theorem 5.14 shows how to convert $g_s$ into a p-value and one would reject the null hypothesis if the p-value is below a pre-specified significance level $\alpha$ (such as 0.01 or 0.05). As a simple experiment to verify the validity of this procedure, we simulate the utility scores of 100 queries by sampling 100 numbers from the datasets and we vary $\epsilon$ from $1 \times 10^{-6}$ to $11 \times 10^{-6}$ to ensure a decent chance of a non-optimal query being returned. We run the Exponential Mechanism with Gap for $n = 100000$ times and record as $c_1$ the number of times the returned $\omega_s$ is not optimal ($H_0$ is true). Thus $c_1/n$ is an estimate of $P(H_0)$. Among the $c_1$ occurrences where $H_0$ is true, we record as $c_2$ the number of times Theorem 5.14 gives a p-value $\leq \alpha$ (for $\alpha = 0.05$ and for $\alpha = 0.01$), causing the null hypothesis to be erroneously rejected. The quantity $c_2/c_1$ is an estimate of how frequently this happens (this is called the Type I error and must be $\leq \alpha$ in order for the hypothesis testing framework to be considered valid). As shown in Figure 5-9, the errors of the hypothesis test using Theorem 5.14 are indeed less than the significance levels.

![Figure 5-9: The estimated probability of $p \leq \alpha$ when the output index from Exponential Mechanism with Gap is not optimal. Utility scores are sampled from BMS-POS.](image-url)
5.9 General Randomness Alignment and Proof of Lemma 5.1

In this section, we prove Lemma 5.1, which was used to establish the privacy properties of the algorithms we proposed. The proof of the lemma requires a more general theorem for working with randomness alignment functions. We explicitly list all of the conditions needed for the sake of reference (many prior works had incorrect proofs because they did not have such a list to follow). In the general setting, the method of randomness alignment requires the following steps.

1. For each pair of adjacent databases $D \sim D'$ and $\omega \in \Omega$, define a randomness alignment $\phi_{D,D'}$ or local alignment functions $\phi_{D,D',\omega} : \mathcal{H}_{D,\omega} \to \mathcal{H}_{D',\omega}$ (see notation in Table 5-2). In the case of local alignments this involves proving that if $M(D, H) = \omega$ then $M(D', \phi_{D,D',\omega}(H)) = \omega$.

2. Show that $\phi_{D,D'}$ (or all the $\phi_{D,D',\omega}$) is one-to-one (it does not need to be onto). That is, if we know $D, D', \omega$ and we are given the value $\phi_{D,D'}(H)$ (or $\phi_{D,D',\omega}(H)$), we can obtain the value $H$.

3. For each pair of adjacent databases $D \sim D'$, bound the alignment cost of $\phi_{D,D'}$ ($\phi_{D,D'}$ is either given or constructed by piecing together the local alignments). Bounding the alignment cost means the following: If $f$ is the density (or probability mass) function of $H$, find a constant $a$ such that $\frac{f(H)}{f(\phi_{D,D'}(H))} \leq a$ for all $H$ (except a set of measure 0). In the case of local alignments, one can instead show the following. For all $\omega$, and adjacent $D \sim D'$ the ratio $\frac{f(H)}{f(\phi_{D,D',\omega}(H))} \leq a$ for all $H$ (except on a set of measure 0).

4. Bound the change-of-variables cost of $\phi_{D,D'}$ (only necessary when $H$ is not discrete). One must show that the Jacobian of $\phi_{D,D'}$, defined as $J_{\phi_{D,D'}} = \frac{\partial \phi_{D,D'}}{\partial H}$, exists (i.e. $\phi_{D,D'}$ is differentiable) and is continuous except on a set of measure 0. Furthermore, for all pairs $D \sim D'$, show the quantity $|\det J_{\phi_{D,D'}}|$ is lower bounded by some constant $b > 0$. If $\phi_{D,D'}$ is constructed by piecing together local alignments $\phi_{D,D',\omega}$ then this is equivalent to showing the following (i) $|\det J_{\phi_{D,D',\omega}}|$ is lower bounded by some constant $b > 0$ for every $D \sim D'$ and $\omega$; and (ii) for each $D \sim D'$, the set $\Omega$ can be partitioned into countably many disjoint measurable sets $\Omega = \bigcup \Omega_i$ such that whenever $\omega$ and $\omega^*$ are in the same partition, then $\phi_{D,D',\omega}$ and $\phi_{D,D',\omega^*}$ are the same function. Note that this last condition (ii) is equivalent to requiring that the local alignments must be defined without using the axiom of choice (since non-measurable sets are not constructible otherwise) and for each $D \sim D'$, the number
of distinct local alignments is countable. That is, the set \( \{ \phi_{D,Y,\omega} \mid \omega \in \Omega \} \) is countable (i.e., for many choices of \( \omega \) we get the same exact alignment function).

**Theorem 5.15.** Let \( M \) be a randomized algorithm that terminates with probability 1 and suppose the number of random variables used by \( M \) can be determined from its output. If, for all pairs of adjacent databases \( D \sim D' \), there exist randomness alignment functions \( \phi_{D,Y} \) (or local alignment functions \( \phi_{D,Y,\omega} \) for all \( \omega \in \Omega \) and \( D \sim D' \)) that satisfy conditions 1 though 4 above, then \( M \) satisfies \( \ln(a/b) \)-differential privacy.

**Proof.** We need to show that for all \( D \sim D' \) and \( E \subseteq \Omega \), \( P(\mathcal{H}_{D:E}) \leq (a/b)P(\mathcal{H}_{D':E}) \).

First we note that if we have a randomness alignment \( \phi_{D,Y} \), we can define corresponding local alignment functions as follows \( \phi_{D,Y,\omega}(H) = \phi_{D,Y}(H) \) (in other words, they are all the same). The conditions on local alignments are a superset of the conditions on randomness alignments, so for the rest of the proof we work with the \( \phi_{D,Y,\omega} \).

Let \( \phi_1, \phi_2, \ldots \) be the distinct local alignment functions (there are countably many of them by Condition 4). Let \( E_i = \{ \omega \in E \mid \phi_{D,Y,\omega} = \phi_i \} \). By Conditions 1 and 2 we have that for each \( \omega \in E_i \), \( \phi_i \) is one-to-one on \( \mathcal{H}_{D:Y,\omega} \) and \( \phi_i(\mathcal{H}_{D:Y,\omega}) \subseteq \mathcal{H}_{D':Y,\omega} \). Note that \( \mathcal{H}_{D:E_i} = \bigcup_{\omega \in E_i} \mathcal{H}_{D:Y,\omega} \) and \( \mathcal{H}_{D':E_i} = \bigcup_{\omega \in E_i} \mathcal{H}_{D':Y,\omega} \). Furthermore, the sets \( \mathcal{H}_{D:Y,\omega} \) are pairwise disjoint for different \( \omega \) and the sets \( \mathcal{H}_{D':Y,\omega} \) are pairwise disjoint for different \( \omega \). It follows that \( \phi_i \) is one-to-one on \( \mathcal{H}_{D:E_i} \) and \( \phi_i(\mathcal{H}_{D:E_i}) \subseteq \mathcal{H}_{D':E_i} \). Thus for any \( H' \in \phi_i(\mathcal{H}_{D:E_i}) \) there exists \( H \in \mathcal{H}_{D:E_i} \) such that \( H = \phi_i^{-1}(H') \). By Conditions 3 and 4, we have

\[
\frac{f(H)}{f(\phi_i(H))} = \frac{f(\phi_i^{-1}(H'))}{f(H')} \leq a \quad \text{for all } H \in \mathcal{H}_{D:E_i}, \quad |\det J_{\phi_i}| \geq b \quad \text{(except on a set of measure 0)}.
\]

Then the following is true:

\[
P(\mathcal{H}_{D:E_i}) = \int_{\mathcal{H}_{D:E_i}} f(H) dH = \int_{\phi_i(\mathcal{H}_{D:E_i})} f(\phi_i^{-1}(H')) \frac{dH'}{|\det J_{\phi_i}|} \leq \int_{\phi_i(\mathcal{H}_{D:E_i})} af(H') \frac{1}{b} dH' = \frac{a}{b} \int_{\phi_i(\mathcal{H}_{D:E_i})} f(H') dH' \leq \frac{a}{b} \int_{\mathcal{H}_{D':E_i}} f(H') dH' = \frac{a}{b} P(\mathcal{H}_{D':E_i}).
\]

The second equation is the change of variables formula in calculus. The last inequality follows from the containment \( \phi_i(\mathcal{H}_{D:E_i}) \subseteq \mathcal{H}_{D':E_i} \) and the fact that the density \( f \) is nonnegative. In the case that \( H \) is discrete, simply replace the density \( f \) with a probability mass function, change the integral into a
summation, ignore the Jacobian term and set $b = 1$. Finally, since $E = \cup_i E_i$ and $E_i \cap E_j = \emptyset$ for $i \neq j$, we conclude that
\[
P(H_{D:E}) = \sum_i P(H_{D:E_i}) \leq \frac{a}{b} \sum_i P(H_{D':E_i}) = \frac{a}{b} P(H_{D':E}).
\]

We now present the proof of Lemma 5.1.

**Proof.** Let $\phi_{D,D',\omega}(H) = H' = (\eta_1', \eta_2', \ldots)$. By acyclicity there is some permutation $\pi$ under which $\eta_{\pi(1)} = \eta_{\pi(1)}' - c$ where $c$ is some constant depending on $D \sim D'$ and $\omega$. Thus $\eta_{\pi(1)}$ is uniquely determined by $H'$. Now (as an induction hypothesis) assume $\eta_{\pi(1)}$, $\ldots$, $\eta_{\pi(j-1)}$ are uniquely determined by $H'$ for some $j > 1$, then $\eta_{\pi(j)} = \eta_{\pi(j)}' - \psi_{D,D',\omega}(\eta_{\pi(1)}, \ldots, \eta_{\pi(j-1)})$, so $\eta_{\pi(j)}$ is also uniquely determined by $H'$. Thus by strong induction $H$ is uniquely determined by $H'$, i.e., $\phi_{D,D',\omega}$ is one-to-one.

It is easy to see that with this ordering, $J_{\phi_{D,D',\omega}}$ is an upper triangular matrix with 1’s on the diagonal. Since permuting variables doesn’t change $|\det J_{\phi_{D,D',\omega}}|$, we have $|\det J_{\phi_{D,D',\omega}}| = 1$ since that is the determinant of upper triangular matrices. Furthermore, (recalling the definition of the cost of $\phi_{D,D',\omega}$) we have $\ln \frac{f(H)}{f(\phi_{\omega}(H))} = \sum_i \ln \frac{f_i(\eta_i)}{f_i(\eta_i')} \leq \sum_i \frac{|\eta_i - \eta_i'|}{\alpha_i} \leq \epsilon$. The first inequality follows from Condition 3 of Lemma 5.1 and the second from Condition 4.

5.10 Additional Proofs

5.10.1 Proof of Theorem 5.9

**Proof.** Let $q_1, \ldots, q_k$ be the true answers to the $k$ queries selected by Noisy-Top-K-with-Gap algorithm. Let $\alpha_i$ be the estimate of $q_i$ using Laplace mechanism, and $g_i$ be the estimate of the gap between $q_i$ and $q_{i+1}$ from Noisy-Top-K-with-Gap.

Recall that $\alpha_i = q_i + \xi_i$ and $g_i = q_i + \eta_i - q_{i+1} - \eta_{i+1}$ where $\xi_i$ and $\eta_i$ are independent Laplacian random variables. Assume without loss of generality that $\text{Var}(\xi_i) = \sigma^2$ and $\text{Var}(\eta_i) = \lambda \sigma^2$. Write in vector notation

\[
\begin{bmatrix}
q_1 \\
\vdots \\
q_k
\end{bmatrix},
\begin{bmatrix}
\xi_1 \\
\vdots \\
\xi_k
\end{bmatrix},
\begin{bmatrix}
\eta_1 \\
\vdots \\
\eta_k
\end{bmatrix},
\begin{bmatrix}
\alpha_1 \\
\vdots \\
\alpha_k
\end{bmatrix},
\begin{bmatrix}
g_1 \\
\vdots \\
g_{k-1}
\end{bmatrix},
\]
then \( \alpha = q + \xi \) and \( g = N(q + \eta) \) where

\[
N = \begin{bmatrix}
1 & -1 \\
\vdots & \ddots \\
1 & -1
\end{bmatrix}_{(k-1) \times k}.
\]

Our goal is then to find the best linear unbiased estimate (BLUE) \( \beta \) of \( q \) in terms of \( \alpha \) and \( g \). In other words, we need to find a \( k \times k \) matrix \( X \) and a \( k \times (k - 1) \) matrix \( Y \) such that

\[
\beta = X\alpha + Yg
\]  \hspace{1cm} (5.5)

with \( E(\|\beta - q\|^2) \) as small as possible. Unbiasedness implies that \( \forall q, E(\beta) = Xq +YNq = q \). Therefore \( X +YN = I_k \) and thus

\[
X = I_k -YN.
\]  \hspace{1cm} (5.6)

Plugging this into (5.5), we have \( \beta = (I_k -YN)\alpha + Yg = \alpha - Y(N\alpha - g) \). Recall that \( \alpha = q + \xi \) and \( g = N(q + \eta) \), we have \( N\alpha - g = N(q + \xi - q - \eta) = N(\xi - \eta) \). Thus

\[
\beta = \alpha - YN(\xi - \eta).
\]  \hspace{1cm} (5.7)

Write \( \theta = N(\xi - \eta) \), then we have \( \beta - q = \alpha - q - Y\theta = \xi - Y\theta \). Therefore, finding the BLUE is equivalent to solving the optimization problem \( Y = \arg \min \Phi \) where

\[
\Phi = E(\|\xi - Y\theta\|^2) = E((\xi - Y\theta)^T(\xi - Y\theta))
\]

\[
= E(\xi^T(\xi - Y\theta - \theta^TY\theta + \theta^TY\theta))^T(\xi - Y\theta + \theta^TY\theta)
\]

Taking the partial derivatives of \( \Phi \) w.r.t \( Y \), we have

\[
\frac{\partial \Phi}{\partial Y} = E(0 - \xi \theta^T - \xi \theta^T + Y(\theta \theta^T + \theta \theta^T))
\]
By setting \( \frac{\partial \Phi}{\partial Y} = 0 \) we have \( YE(\theta \theta^T) = E(\xi \theta^T) \) thus

\[
Y = E(\xi \theta^T)E(\theta \theta^T)^{-1}.
\]  

(5.8)

Recall that \( (\xi \theta^T)_{ij} = \xi_i(\xi_j - \xi_{j+1} - \eta_j + \eta_{j+1}) \), we have

\[
E(\xi \theta^T) = \begin{cases} 
E(\xi_i^2) = \text{Var}(\xi_i) = \sigma^2 & i = j \\
-E(\xi_i^2) = -\text{Var}(\xi_i) = -\sigma^2 & i = j + 1 \\
0 & \text{otherwise}
\end{cases}
\]

Hence

\[
E(\xi \theta^T) = \sigma^2 \begin{bmatrix} 
1 & \cdots & \cdots & 1 \\
-1 & \cdots & \cdots & -1 \\
\vdots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
-1 & \cdots & \cdots & -1
\end{bmatrix}_{k \times (k-1)} = \sigma^2 N^T.
\]

Similarly, we have

\[
(\theta \theta^T)_{ij} = (\xi_i - \xi_{i+1} - \eta_i + \eta_{i+1})(\xi_j - \xi_{j+1} - \eta_j + \eta_{j+1}) \\
= \xi_i \xi_j + \xi_{i+1} \xi_{j+1} - \xi_i \xi_{j+1} - \xi_{i+1} \xi_j \\
+ \eta_i \eta_j + \eta_{i+1} \eta_{j+1} - \eta_i \eta_{j+1} - \eta_{i+1} \eta_j \\
- (\xi_i - \xi_{i+1})(\eta_j - \eta_{j+1}) - (\eta_i - \eta_{i+1})(\xi_j - \xi_{j+1})
\]

Thus

\[
E(\theta \theta^T) = \begin{cases} 
E(\xi_i^2 + \xi_{i+1}^2 + \eta_i^2 + \eta_{i+1}^2) = 2(1+\lambda)\sigma^2 & i = j \\
E(-\xi_i^2 - \eta_i^2) = -(1+\lambda)\sigma^2 & i = j + 1 \\
E(-\xi_j^2 - \eta_j^2) = -(1+\lambda)\sigma^2 & i = j - 1 \\
0 & \text{otherwise}
\end{cases}
\]
Hence

\[
E(\theta\theta^T) = (1+\lambda)\sigma^2 \begin{bmatrix}
2 & -1 \\
-1 & 2 & -1 \\
\vdots & \ddots & \ddots \\
-1 & 2 & -1 \\
0 & 0 & \ddots & \ddots \\
\end{bmatrix}_{(k-1)\times(k-1)}
\]

It can be directly computed that \(E(\theta\theta^T)^{-1}\) is a symmetric matrix whose lower triangular part is

\[
\frac{1}{k(1+\lambda)\sigma^2} \begin{bmatrix}
(k-1) \cdot 1 & \cdots & \cdots & \cdots & \cdots \\
(k-2) \cdot 1 & (k-2) \cdot 2 & \cdots & \cdots & \cdots \\
(k-3) \cdot 1 & (k-3) \cdot 2 & (k-3) \cdot 3 & \cdots & \cdots \\
\vdots & \vdots & \vdots & \ddots & \ddots \\
1 \cdot 1 & 1 \cdot 2 & 1 \cdot 3 & \cdots & 1 \cdot (k-1) \\
\end{bmatrix}
\]

i.e., \(E(\theta\theta^T)^{-1}_{ij} = E(\theta\theta^T)^{-1}_{ji} = \frac{1}{k(1+\lambda)\sigma^2} \cdot (k-i) \cdot j\) for all \(1 \leq i \leq j \leq k-1\). Therefore, \(Y = E(\xi\theta^T)E(\theta\theta^T)^{-1} = \frac{1}{k(1+\lambda)\sigma^2} \begin{bmatrix}
k-1 & k-2 & \cdots & 1 \\
k-1 & k-2 & \cdots & 1 \\
k-1 & k-2 & \cdots & 1 \\
\vdots & \vdots & \ddots & \ddots \\
k-1 & k-2 & \cdots & 1 \\
\end{bmatrix} \begin{bmatrix}0 & 0 & \cdots & 0 \\
0 & k & \cdots & 0 \\
k & k & \cdots & 0 \\
\vdots & \vdots & \ddots & 0 \\
k & k & \cdots & k \\
\end{bmatrix}_{k\times(k-1)}\)

Hence

\[
X = I_k - YN = \frac{1}{k(1+\lambda)} \begin{bmatrix}
1+k\lambda & 1 & \cdots & 1 \\
1 & 1+k\lambda & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \cdots & 1+k\lambda \\
\end{bmatrix}_{k\times k}
\]

\(\square\)
5.10.2 Proof of Corollary 5.9.1

Recall that $\alpha_i = q_i + \xi_i$ and $g_i = q_i + \eta_i - q_{i+1} - \eta_{i+1}$ where $\xi_i$ and $\eta_i$ are independent Laplacian random variables. Assume without loss of generality that $\text{Var}(\xi_i) = \sigma^2$ and $\text{Var}(\eta_i) = \lambda \sigma^2$ as before. From the matrices $X$ and $Y$ in Theorem 5.9 we have that $\beta_i = \frac{x_i + y_i}{k(1+\lambda)}$ where

$$x_i = \alpha_1 + \cdots + (1 + k \lambda) \alpha_i + \cdots + \alpha_k$$

$$= (q_1 + \xi_1) + \cdots + (1 + k \lambda) (q_i + \xi_i) + \cdots + (q_k + \xi_k)$$

and

$$y_i = -g_1 - 2g_2 - \cdots - (i-1)g_{i-1} + (k-i)g_i + \cdots + 2g_{k-2} + g_{k-1}$$

$$= -(q_1 + \eta_1) - (q_2 + \eta_2) - \cdots - (q_{i-1} + \eta_{i-1}) + (k-1)(q_i + \eta_i) - (q_{i+1} + \eta_{i+1}) - \cdots - (q_k + \eta_k).$$

Therefore

$$\text{Var}(x_i) = \sigma^2 + \cdots + (1 + k \lambda)^2 \sigma^2 + \cdots + \sigma^2$$

$$= (k^2 \lambda^2 + 2k \lambda + k) \sigma^2$$

$$\text{Var}(y_i) = \lambda \sigma^2 + \cdots + (k-1)^2 \lambda \sigma^2 + \cdots + \lambda \sigma^2$$

$$= (k^2 - k) \lambda \sigma^2$$

and thus $\text{Var}(\beta_i) = \frac{\text{Var}(x_i) + \text{Var}(y_i)}{k^2 (1+\lambda)^2} = \frac{1 + k \lambda}{k^2 (1+\lambda)^2} \sigma^2$. Recall that $\text{Var}(\alpha_i) = \text{Var}(\xi_i) = \sigma^2$, we have $\frac{\text{Var}(\beta_i)}{\text{Var}(\alpha_i)} = \frac{1 + k \lambda}{k^2 (1+\lambda)^2} \cdot \sigma^2$. 


5.10.3 Proof of Lemma 5.4

The density function of $\eta_i - \eta$ is $f_{\eta_i - \eta}(z) = \int_{-\infty}^{\infty} f_{\eta_i}(x) f_{\eta}(x-z) \, dx = \frac{\epsilon_0 \epsilon_*}{4} \int_{-\infty}^{\infty} e^{-\epsilon_* |x|} e^{-\epsilon_0 |x-z|} \, dx$. First consider the case $\epsilon_0 \neq \epsilon_*$. When $z \geq 0$, we have

$$f_{\eta_i - \eta}(z) = \frac{\epsilon_0 \epsilon_*}{4} \int_{-\infty}^{\infty} e^{-\epsilon_* x} e^{\epsilon_0 (x-z)} \, dx + \int_0^\infty e^{-\epsilon_* x} e^{\epsilon_0 (x-z)} \, dx + \int_0^\infty e^{-\epsilon_* x} e^{-\epsilon_0 (x-z)} \, dx$$

$$= \frac{\epsilon_0 \epsilon_*}{4} \left( \frac{e^{-\epsilon_0 z}}{\epsilon_0 + \epsilon_*} + \frac{e^{-\epsilon_* z} - e^{-\epsilon_0 z}}{\epsilon_0 - \epsilon_*} + \frac{e^{-\epsilon_* z}}{\epsilon_0 + \epsilon_*} \right)$$

$$= \frac{\epsilon_0 \epsilon_* (e^{-\epsilon_0 z} - \epsilon_* e^{-\epsilon_* z})}{2(\epsilon_0^2 - \epsilon_*^2)}$$

Thus by symmetry we have that for all $z \in \mathbb{R}$, $f_{\eta_i - \eta}(z) = \frac{\epsilon_0 \epsilon_*(e^{-\epsilon_0 z} - \epsilon_* e^{-\epsilon_* z})}{2(\epsilon_0^2 - \epsilon_*^2)}$, and

$$\mathbb{P}(\eta_i - \eta \geq -t) = \int_{-t}^{\infty} f_{\eta_i - \eta}(z) \, dz = \int_{-t}^{0} f_{\eta_i - \eta}(z) \, dz + \frac{1}{2}$$

$$= 1 - \frac{\epsilon_0^2 e^{-\epsilon_* t} - \epsilon_*^2 e^{-\epsilon_0 t}}{2(\epsilon_0^2 - \epsilon_*^2)}.$$ 

Now if $\epsilon_0 = \epsilon_*$, by similar computations we have $f_{\eta_i - \eta}(z) = (\frac{\epsilon_0}{4} + \frac{\epsilon_*^2 |z|}{4}) e^{-\epsilon_0 |z|}$ and $\mathbb{P}(\eta_i - \eta \geq -t) = 1 - \left( \frac{2 \epsilon_0 t}{4} \right) e^{-\epsilon_0 t}$.

5.10.4 Proofs in Section 5.7

A well-known, but inefficient, folklore algorithm for the Exponential Mechanism is based on the Gumbel Max Trick [94, 95]: given numbers $\mu_1, \ldots, \mu_n$, add independent Gumbel(0) noise to each and select the index of the largest noisy value. This is the same as sampling the $i^{th}$ item with probability proportional to $e^{\mu_i}$. Let $\text{Cat}(\mu_1, \ldots, \mu_n)$ denote the categorical distribution that returns item $\omega_i$ with probability $\frac{\exp(\mu_i)}{\sum_{j=1}^{n} \exp(\mu_j)}$. The Gumbel-Max theorem provides distributions for the identity of the noisy maximum and the value of the noisy maximum:
Theorem 5.16 (The Gumbel-Max Trick [94, 95]). Let \( G_i, \ldots, G_n \) be i.i.d. Gumbel(0) random variables and let \( \mu_1, \ldots, \mu_n \) be real numbers. Define \( X_i = G_i + \mu_i \). Then

1. The distribution of \( \arg \max_i (X_1, \ldots, X_n) \) is the same as \( \text{Cat}(\mu_1, \ldots, \mu_n) \).

2. The distribution of \( \max_i (X_1, \ldots, X_n) \) is the same as the Gumbel\((\ln \sum_{i=1}^n \exp(\mu_i))\) distribution.

Using the Gumbel-Max trick, one can propose an Exponential Mechanism with Gap by replacing Laplace or Exponential noise in Noisy Max with Gap with the Gumbel distribution as shown in Algorithm 22 (boxed items represent gap information). We first prove the correctness of this algorithm and then show how to replace the Gumbel max trick with any efficient black box algorithm for the exponential mechanism.

Algorithm 22: Exponential Mechanism w. Gap

\[
\begin{align*}
\text{input} & : \mu: \text{utility function with sensitivity } \Delta_\mu \\
& \quad D: \text{database, } \epsilon: \text{privacy budget} \\
\text{function} & \text{GapExpMech } (D, \mu, \epsilon): \\
& \quad \text{foreach } i \in \{1, \ldots, n\} \text{ do} \\
& \quad \quad x_i \leftarrow \epsilon \mu(D, \omega_i)/2 \Delta_\mu + \text{Gumbel}(0) \\
& \quad s, t \leftarrow \text{arg max}_2(x_1, \ldots, x_n) \\
& \quad \text{return } \omega_s, x_s - x_t
\end{align*}
\]

We first need the following results.

Lemma 5.17. Let \( \epsilon > 0 \). Let \( \mu : D \times R \rightarrow R \) be a utility function of sensitivity \( \Delta_\mu \). Define \( \nu : D \rightarrow R \) and its sensitivity \( \Delta_\nu \) as

\[
\nu(D) = \ln \sum_{\omega \in R} e^{\epsilon \mu(D, \omega) / 2 \Delta_\mu}, \quad \Delta_\nu = \max_{D, D'} |\nu(D) - \nu(D')|.
\]

Then \( \Delta_\nu \), the sensitivity of \( \nu \), is at most \( \frac{\epsilon}{2} \).

Proof of Lemma 5.17. From the definition of \( \nu \) we have

\[
|\nu(D) - \nu(D')| = |\ln \sum_{\omega \in R} e^{\epsilon \mu(D, \omega) / 2 \Delta_\mu} - \ln \sum_{\omega \in R} e^{\epsilon \mu(D', \omega) / 2 \Delta_\mu}| \\
= |\ln \left( \sum_{\omega \in R} e^{\epsilon \mu(D, \omega) / 2 \Delta_\mu} \right) / \left( \sum_{\omega \in R} e^{\epsilon \mu(D', \omega) / 2 \Delta_\mu} \right)|
\]
By definition of sensitivity, we have

$$\mu(D', \omega) - \Delta \mu \leq \mu(D, \omega) \leq \mu(D', \omega) + \Delta \mu,$$

and therefore

$$e^{-\frac{\Delta}{2\Delta \mu}} \sum_{\omega \in \mathcal{R}} e^{\frac{\mu(D', \omega)}{2\Delta \mu}} \leq \sum_{\omega \in \mathcal{R}} e^{\frac{\mu(D, \omega)}{2\Delta \mu}} \leq e^{\frac{\Delta}{2\Delta \mu}} \sum_{\omega \in \mathcal{R}} e^{\frac{\mu(D', \omega)}{2\Delta \mu}}.$$

Thus $|\nu(D) - \nu(D')| \leq \frac{\xi}{2}$, and hence $\Delta \nu \leq \frac{\xi}{2}$. \hfill \qed

**Lemma 5.18.** Let $f(x; \theta) = \frac{e^{-(x - \theta)}}{(1 + e^{-(x - \theta)})^2}$ be the density of the logistic distribution, then $|\ln \frac{f(x; \theta)}{f(x; \theta')}| \leq |\theta - \theta'|$.

**Proof of Lemma 5.18.** Note that $|\ln \frac{f(x; \theta)}{f(x; \theta')}| = |\ln \frac{f(x; \theta')}{f(x; \theta)}|$ so without loss of generality, we can assume that $\theta \geq \theta'$ (i.e., the location parameter in the numerator is $\geq$ the parameter in the denominator). From the formula of $f$ we have $\frac{f(x; \theta)}{f(x; \theta')} = e^{\theta - \theta'} \cdot \left(\frac{1 + e^{-x} e^{\theta'}}{1 + e^{-x} e^{\theta}}\right)^2$. Clearly $e^{\theta} \geq e^{\theta'} \Rightarrow \frac{1 + e^{-x} e^{\theta'}}{1 + e^{-x} e^{\theta}} \leq 1$. Also,

$$1 + e^{-x} e^{\theta'} = e^{\theta - \theta'} \cdot \frac{e^{\theta - \theta'} + e^{-x} e^{\theta}}{1 + e^{-x} e^{\theta}} \geq \frac{e^{\theta - \theta'} (1 + e^{-x} e^{\theta})}{1 + e^{-x} e^{\theta}} = e^{\theta - \theta'}.$$

Therefore, $e^{\theta - \theta'} \cdot (e^{\theta - \theta'} - 1)^2 \leq \frac{f(x; \theta)}{f(x; \theta')} \leq e^{\theta - \theta'}$. Thus $|\ln \frac{f(x; \theta)}{f(x; \theta')}| \leq |\theta - \theta'|$. \hfill \qed

**Theorem 5.19.** Algorithm 22 satisfies $\epsilon$-differential privacy. Its output distribution is equivalent to selecting $\omega_s$ with probability proportional to $\exp\left(\frac{\epsilon \mu(D, \omega_s)}{2\Delta \mu}\right)$ and then independently sampling the gap from the Logistic distribution (conditional on only sampling non-negative values) with location parameter $\theta = \frac{\epsilon \mu(D, \omega_s)}{2\Delta \mu} - \ln \sum_{j \neq s} \exp\left(\frac{\epsilon \mu(D, \omega_j)}{2\Delta \mu}\right)$.

**Proof of Theorem 5.19.** For $\omega_i \in \mathcal{R}$, let $\mu_i = \frac{\epsilon \mu(D, \omega_s)}{2\Delta \mu}$ and $\mu'_i = \frac{\epsilon \mu(D', \omega_s)}{2\Delta \mu}$. Let $X_i \sim \text{Gumbel}(\mu_i)$ and $X'_i \sim \text{Gumbel}(\mu'_i)$.

We consider the probability of outputting the selected $\omega_s$ with gap $\gamma \geq 0$ when $D$ is the input database:

$$P(\omega_s \text{ is chosen with gap } \geq \gamma \mid D) = \int_{\mathbb{R}} P(X_s = z + \gamma) \prod_{i \neq s} P(X_i \leq z) \, dz$$
\[ \begin{align*} 
\text{let } \mu & \text{ has value } \mu_s \text{ and so equal to } \\
\text{in Equation 5.10, the term } e^{\mu_s-\gamma} & \text{ is the density of the event that a logistic random variable with location } \\
& \text{ is chosen with gap } s \text{ and is nonnegative.}
\end{align*} \]

\[ P(\omega_s \text{ is chosen with gap } \in [0, \gamma] | D) = \frac{e^{\mu_s}}{e^{\mu_s} + e^{\mu^*}} - \frac{1}{1 + e^{-(\mu_s-\gamma-\mu^*)}} = \frac{1}{1 + e^{-\theta}} - \frac{1}{1 + e^{-(\theta-\gamma)}} \]

Taking the derivative with respect to \( \gamma \), we get the density \( f(\omega_s, \gamma | D) \) of \( \omega_s \) being chosen with gap equal to \( \gamma \):

\[ f(\omega_s, \gamma | D) = \frac{d}{d\gamma} \left( \frac{1}{1 + e^{-\theta}} - \frac{1}{1 + e^{-(\theta-\gamma)}} \right) \]

\[ = \frac{e^{-(\gamma-\theta)}}{(e^{-(\gamma-\theta)} + 1)^2} \cdot 1_{[\gamma \geq 0]} \]

\[ = \frac{e^{\mu_s}}{e^{\mu_s} + e^{\mu^*}} \left( \frac{e^{-(\gamma-\theta)}}{(e^{-(\gamma-\theta)} + 1)^2} \cdot 1_{[\gamma \geq 0]} \right) / \frac{e^{\mu_s}}{e^{\mu_s} + e^{\mu^*}} \]

\[ = \frac{e^{\mu_s}}{e^{\mu_s} + e^{\mu^*}} \left( \frac{e^{-(\gamma-\theta)}}{(e^{-(\gamma-\theta)} + 1)^2} \cdot 1_{[\gamma \geq 0]} \right) / 1 + e^{-\theta} \]

(5.10)
Finally, the term \( \frac{1}{1+e^{-\theta}} \) is the probability that a logistic random variable with location \( \theta \) is nonnegative.

Thus \( \left( \frac{e^{-x-\theta}}{(e^{-x-\theta}+1)^2} \right) I_{\{x \geq 0\}} \left/ \frac{1}{1+e^{-\theta}} \right. \) is the probability of a logistic random variable having value \( \gamma \) conditioned on it being nonnegative.

Therefore Equation 5.10 is the probability of selecting \( \omega_s \) and independently sampling a nonnegative value \( \gamma \) from the conditional logistic distribution location parameter \( \theta = \mu_s - \mu^* \) (i.e., conditional on it only returning nonnegative values).

Now, recall that \( \mu_i = \frac{\mu_i(D,i)}{2\Delta \mu} \), we apply Lemmas 5.18 and 5.17 with the help of Equation 5.9 to finish the proof:

\[
| \ln f(\omega_s, \gamma | D) \left/ f(\omega_s, \gamma | D') \right. | \leq |(\mu_s - \mu^*) - (\mu'_s - \mu''^*)| \\
\leq |\mu_s - \mu'_s| + |\ln \sum_{i \neq s} e^{\mu_i} - \ln \sum_{i \neq s} e^{\mu'_i}| \leq \varepsilon/2 + \varepsilon/2 = \varepsilon.
\]

\[\square\]

**Proof of Theorem 5.13.** The first part follows directly from Theorem 5.19. Also, from the proof of Theorem 5.19 the gap \( g_s \) has density \( f(x; \theta) = \frac{e^{-x-\theta}}{(e^{-x-\theta}+1)^2} I_{\{x \geq 0\}} \left/ \frac{1}{1+e^{-\theta}} \right. \). Since

\[
\int_0^t e^{-x+\theta} \left/ (e^{-x+\theta}+1)^2 \right. \cdot x \, dx = \int_0^t \frac{e^{x-\theta}}{(1+e^{x-\theta})^2} \cdot x \, dx \\
= \int_0^t x \cdot \left( \frac{-1}{1+e^{x-\theta}} \right)' \, dx = \frac{-x}{1+e^{x-\theta}} \bigg|_0^t + \int_0^t \frac{1}{1+e^{x-\theta}} \, dx \\
= \frac{-t}{1+e^{t-\theta}} + (x - \ln(1+e^{x-\theta})) \bigg|_0^t \\
= \frac{-t}{1+e^{t-\theta}} + t - \ln(1+e^{t-\theta}) + \ln(1+e^{x-\theta}) \\
= \frac{-t}{1+e^{t-\theta}} + \ln \frac{e^t}{1+e^{t-\theta}} + \ln(1+e^{-\theta})
\]

We have

\[
\int_0^\infty \frac{e^{-x+\theta}}{(e^{-x+\theta}+1)^2} \cdot x \, dx = \lim_{t \to \infty} \int_0^t \frac{e^{-x+\theta}}{(e^{-x+\theta}+1)^2} \cdot x \, dx \\
= \lim_{t \to \infty} \left( \frac{-t}{1+e^{t-\theta}} + \ln \frac{e^t}{1+e^{t-\theta}} + \ln(1+e^{-\theta}) \right) \\
= 0 + \ln(e^\theta) + \ln(1+e^{-\theta}) = \ln(1+e^\theta)
\]
Hence $\mathbb{E}(g_s) = (1 + e^{-\theta}) \ln(1 + e^\theta)$.

□

Proof of Theorem 5.14. Assume $H_0$ is true, i.e., there exists a $t \neq s$ such that $\mu(D, \omega_s) < \mu(D, \omega_t)$. Then

$$\theta = \frac{\epsilon \mu(D, \omega_s)}{2 \Delta \mu} - \ln \sum_{j \neq s} \exp \left( \frac{\epsilon \mu(D, \omega_j)}{2 \Delta \mu} \right)$$

$$\leq \frac{\epsilon \mu(D, \omega_s)}{2 \Delta \mu} - \ln \exp \frac{\epsilon \mu(D, \omega_t)}{2 \Delta \mu} = \frac{\epsilon \mu(D, \omega_s)}{2 \Delta \mu} - \frac{\epsilon \mu(D, \omega_t)}{2 \Delta \mu} < 0$$

Using be the density of the gap from above, we have

$$P(x \geq \gamma \mid H_0) = (1 + e^{-\theta}) \int_{\gamma}^{\infty} \frac{e^{-x+\theta}}{(1 + e^{-x+\theta})^2} \, dx$$

$$= (1 + e^{-\theta}) \int_{\gamma}^{\infty} \frac{e^{x-\theta}}{(1 + e^{x-\theta})^2} \, dx = (1 + e^{-\theta}) \cdot \left( \frac{-1}{1 + e^{x-\theta}} \bigg|_{\gamma}^{\infty} \right)$$

$$= \frac{1 + e^{-\theta}}{1 + e^{\gamma-\theta}} = \frac{e^\theta + 1}{e^\theta + e^\gamma} < \frac{2}{1 + e^\gamma}$$

because $\frac{e^\theta + 1}{e^\theta + e^\gamma}$ is an increasing function of $\theta$ and $\theta < 0$. □
Chapter 6

Secure Implementation of Private Selection Mechanisms

6.1 Introduction

Because of the statistical nature of this privacy definition, most differentially private algorithms introduce noise from continuous probability distributions (e.g., the Laplace distribution) to provide privacy. However, these distributions cannot be faithfully represented, let alone sampled from, on physical machines which use only finite precision approximations to real number arithmetic, e.g., via floating-point numbers.

One might think that such issues are purely of theoretical interest and do not cause serious harm in practice. Unfortunately, this is not the case: Mironov [23] demonstrated that textbook implementations of the Laplace mechanism, the most basic algorithm to satisfy differential privacy, can lead to catastrophic failures of privacy. In particular, by examining the low-order bits of the noisy output, the noiseless value can often be determined. Mironov demonstrated that this information allows an entire dataset of 18K records to be rapidly reconstructed with a negligible ($< 10^{-6}$) total privacy budget. Despite this demonstration, the flawed methods continue to appear in open source implementations of differentially private mechanisms. This shows a real need for algorithm designers to provide secure and practical solutions to enable the deployment of differentially private systems in privacy-critical settings.

An important class of mechanisms in the differential privacy toolbox is private selection. For example, these mechanisms serve as key components in many privacy preserving algorithms for synthetic data generation [59], ordered statistics [96], quantiles [97], frequent itemset mining [61], hyperparameter tuning [58] for statistical models, etc. In Chapter 5, we presented novel variants of these mechanisms that provide more functionality at the same privacy cost (under pure differential privacy). However, these variants are susceptible to attacks that exploit the floating-point vulnerability because their outputs contain real values that depend on noises sampled from continuous distributions.
In this work, we carefully consider how to securely implement these selection mechanisms on finite computers that cannot faithfully represent real numbers. Our contribution is a secure implementation of the Noisy Top-k with Gap algorithm, which uses sampling primitives based on integers and thus are immune to the aforementioned attacks.

The rest of the chapter is organized as follows. We discuss relevant background in Section 6.2. We present our algorithms in Section 6.3 and proofs for their correctness in Section 6.4.

6.2 Background

6.2.1 Floating Point Vulnerability

One of the most common methods to sample a random value $X$ from a distribution with cumulative distribution function (CDF) $F(\cdot)$ is the inverse sampling method: draw a sample $U$ from the uniform distribution on $[0, 1)$ and apply the inverse CDF to obtain $X = F^{-1}(U)$. It’s easy to check that the CDF of $X$ is indeed $F(\cdot)$:

\[
\forall x, P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x).
\]

The inverse CDF of Laplace and exponential distributions are particularly simple. Thus most software libraries use this method to sample from these (and many other) distributions.

However, the Laplace and exponential distributions are continuous over the real numbers. As such, it is not possible to even represent a sample from them on a finite computer, much less to produce one. On one hand, given the non-uniform density of floating-point numbers, a uniform distribution over $[0, 1)$ is not well-defined. On the other hand, floating-point operations involved in applying the inverse CDF will result in missing values and values that appear more frequently than they should [23].

Mironov [23] demonstrated that textbook implementations of differential privacy mechanisms can lead to catastrophic failures of privacy. In particular, by examining the low-order bits of the outputs of the Laplace mechanism, the true value can often be recovered easily.
6.3 Algorithms

6.3.1 Notation and Setup

In this and the next section, we use the following notation. Let $\gamma$ be a base constant that is floating-point representable (e.g., a negative power of 2). For positive integers $k$ we use $[k]$ to denote the set of integers from 1 to $k$. We use $\pi$ to denote a permutation on $[k]$, i.e., an injective function $\pi : [k] \rightarrow [k]$. For a real number $x \in \mathbb{R}$, we use $\lfloor x \rfloor$ to denote the floor of $x$, i.e., the largest integer $\leq x$. We use $\lfloor x \rfloor_\gamma$ to denote the largest multiple of $\gamma$ that is $\leq x$, which can be expressed as $\lfloor x \rfloor_\gamma = \lfloor \frac{x}{\gamma} \rfloor \cdot \gamma$. This notation is summarized in Table 6-1.

Table 6-1: Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lfloor k \rfloor$</td>
<td>${1, 2, \ldots, k}$</td>
</tr>
<tr>
<td>$\pi$</td>
<td>permutation on $[k]$ (injective function $[k] \rightarrow [k]$)</td>
</tr>
<tr>
<td>$\lfloor x \rfloor$</td>
<td>the largest integer $\leq x$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>floating-point representable base (e.g., a negative power of 2)</td>
</tr>
<tr>
<td>$\lfloor x \rfloor_\gamma$</td>
<td>rounding of $x$ down to the nearest multiple of $\gamma$, $\lfloor x \rfloor_\gamma = \lfloor \frac{x}{\gamma} \rfloor \cdot \gamma$</td>
</tr>
</tbody>
</table>

We will use the exponential distribution and the geometric distribution in our algorithms and proofs. For consistency we use $X$ (resp. $Y$) to denote a random variable following the exponential (resp. geometric) distribution. We use $X'$ (resp. $Y'$) to denote a random variable following the truncated exponential (resp. truncated geometric) distribution. Their basic properties are listed in Table 6-2.

Table 6-2: Noise Distributions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Support</th>
<th>Density/Mass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \sim \text{Exp}(\beta)$</td>
<td>$[0, \infty)$</td>
<td>$\frac{1}{\beta} e^{-\frac{x}{\beta}}$</td>
<td>Exponential dist.</td>
</tr>
<tr>
<td>$X' \sim \text{Exp}(\beta) \mod \gamma$</td>
<td>$[0, \gamma)$</td>
<td>$\frac{1}{\beta} e^{-\frac{x}{\beta}} \big/ \left(1 - e^{-\frac{\gamma}{\beta}}\right)$</td>
<td>Truncated exponential dist.</td>
</tr>
<tr>
<td>$Y \sim \text{Geo}(p)$</td>
<td>${0, 1, \ldots}$</td>
<td>$p (1 - p)^m$</td>
<td>Geometric dist.</td>
</tr>
<tr>
<td>$Y' \sim \text{Geo}(p) \mod M$</td>
<td>${0, 1, \ldots, M-1}$</td>
<td>$\frac{p (1-p)^m}{1-(1-p)^M}$</td>
<td>Truncated geometric dist.</td>
</tr>
</tbody>
</table>

Most real-world applications require the value of real numbers to be accurate to a target precision $\gamma_*$. Without loss of generality, we can assume $\gamma_*$ is $2^{-t}$ for some $t \geq 0$ because we can refine any precision to
its nearest power of 2. Furthermore, rounding query answers does not change the sensitivity of the query.

**Lemma 6.1.** Suppose \( q \) is a scalar-valued function with \( \ell_1 \) sensitivity \( \Delta \). Suppose that \( \gamma \) is a base and that \( \Delta \) is an integer multiple of \( \gamma \). Then \([q]_\gamma\) has sensitivity \( \Delta \).

**Proof.** Without loss of generality, assume \( q(D) \geq q(D') \). Then \( q(D) - \Delta + k\gamma + s = q(D') \) for some nonnegative integer \( k \) (where \( k \leq \Delta/\gamma + 1 \)) and \( s \in [0, \gamma) \). Let \( \text{rem}(q(D)) = q(D) - [q(D)]_\gamma \). Then

\[
[q(D)]_\gamma - \Delta + k\gamma + \text{rem}(q(D)) + s = [q(D')]_\gamma + \text{rem}(q(D'))
\]

Noting that \( \text{rem}(q(D)) + s \in [0, 2\gamma) \), this means

\[
[q(D)]_\gamma - \Delta + k'\gamma + s' = [q(D')]_\gamma + \text{rem}(q(D'))
\]

where \( k' \in \{k, k + 1\} \) and \( s' \in [0, \gamma) \). This means that \( s' = \text{rem}(q(D')) \) as both quantities are nonnegative and less than \( \gamma \), while everything else is a nonnegative multiple of \( \gamma \). Rounding both sides down to the nearest multiple of \( \gamma \), we get

\[
[q(D)]_\gamma - [q(D')]_\gamma = \Delta - k'\gamma \in [-\gamma, \Delta] \subseteq [-\Delta, \Delta]
\]

since \( \Delta \) is a multiple of \( \gamma \). \( \square \)

Thus without loss of generality, we can assume all query answers are multiples of our target base \( \gamma' \), since we can always round them down.
6.3.2 Secure Primitives

We use the following sampling primitives to implement our algorithms. These sampling algorithms do not use floating-point operations and are thus free of the floating-point vulnerability.

Random integer generation from an interval In simulations, probabilistic algorithms and statistical tests, we often generate random integers in a finite range. Pseudo-random values are usually generated in words of a fixed number of bits (e.g., 32 bits, 64 bits) using algorithms such as Mersenne Twister [98], Xorshift [99, 100] and linear congruential generators [101, 102, 103, 104]. To generate a random integer uniformly from an interval (e.g. \([0, k)\)), we need functions to convert such random words to integers in this range without introducing statistical biases.

A straightforward way to achieve this result is by rejection sampling [105]. For example, we can keep generating 32-bit random words until we get an integer less than \(k\). However, this method is inefficient when rejection happens frequently. Popular software libraries use more efficient algorithms [106, 107, 108] to solve this problem. Therefore, we assume access to fast random integer generation is available.

The Fisher-Yates random shuffle Another commonly used technique is the Fisher-Yates shuffle [109, 110]. It randomly permutes a list of \(n\) elements so that all \(n!\) possible permutations are equally probable. This algorithm is based on uniform sampling of integers and its detail is given in Algorithm 23.

<table>
<thead>
<tr>
<th>Algorithm 23: The Fisher-Yates Shuffle: it permutes a list of (n) elements so that all (n!) possible permutations are equiprobable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong>: a list of (n) elements (a_1, \ldots, a_n)</td>
</tr>
<tr>
<td><strong>output</strong>: the same list with all values randomly permuted</td>
</tr>
<tr>
<td><strong>function</strong> Shuffle((a_1, \ldots, a_n)):</td>
</tr>
<tr>
<td>1. <strong>for</strong> (i) from (n) <strong>down to</strong> 2 <strong>do</strong></td>
</tr>
<tr>
<td>2. (j \leftarrow U{1, \ldots, i})</td>
</tr>
<tr>
<td>3. (\text{tmp} \leftarrow a_i) // Swap (a_j) and (a_i)</td>
</tr>
<tr>
<td>4. (a_i \leftarrow a_j)</td>
</tr>
<tr>
<td>5. (a_j \leftarrow \text{tmp})</td>
</tr>
<tr>
<td>6. <strong>return</strong> (a_1, \ldots, a_n)</td>
</tr>
</tbody>
</table>

Sampling from a Bernoulli distribution For certain values of the success probability parameter \(p\), it is possible to reduce sampling from \(\text{Bernoulli}(p)\) to sampling random integers. First, when \(p = n/d\) is a
rational number, we can use the rejection method [105] again. To sample Bernoulli(\(n/d\)) it suffices to draw an integer \(N\) uniformly from the range \([0, d]\) and output 1 if \(N < n\) and 0 otherwise. When \(p = e^{-\theta}\) for some positive rational number \(\theta \in \mathbb{Q}^+\), Cannonne et al. [72] reduced the task of sampling from Bernoulli(\(e^{-\theta}\)) to that of sampling from Bernoulli(\(\theta/k\)) for various integers \(k \geq 1\). For completeness we include their algorithm as Algorithm 24 below.

**Algorithm 24: Sampling from a Bernoulli Distribution**

```
function Bernoulli(e^{-\theta}):
    if \(\theta \in [0, 1]\) then
        K ← 1
        while true do
            A ← Bernoulli(\(\theta/K\))
            if A = 0 then
                break
            else
                K ← K + 1
        if K is odd then
            return 1
        else
            return 0
    else
        for k = 1 to \([\theta]\) do
            B ← Bernoulli(e^{-1}) // Recursive call.
            if B = 0 then
                return 0
            C ← Bernoulli(e^{-(\theta-[\theta])}) // Recursive call. \(\theta-[\theta] \in [0,1)\)
        return C
```

**Sampling from a geometric distribution** Cannonne et al. [72] also showed that sampling from a Geo\((1 - e^{-\theta})\) distribution where \(\theta\) is a positive rational number can be reduced to sampling from Bernoulli\((e^{-\theta})\). The detail is given in Algorithm 25.
Algorithm 25: Sampling from a Geometric Distribution

### input:
Parameters $s, t \geq 1$

### output:
One sample from $\text{Geo}(1 - e^{-s/t})$

```plaintext
function Geometric($1 - e^{-s/t}$):

1. $D \leftarrow 0$
2. while $D = 0$ do
3.   $U \leftarrow U\{0, \ldots, t - 1\}$
4.   $D \leftarrow \text{Bernoulli}(e^{-U/t})$  // Use Algorithm 24
5. $V \leftarrow 0$  // Generate $V$ from $\text{Geo}(1 - e^{-1})$
6. while true do
7.     $A \leftarrow \text{Bernoulli}(e^{-1})$  // Use Algorithm 24
8.     if $A = 0$ then
9.         Break
10.    else
11.       $V \leftarrow V + 1$
12. $X \leftarrow U + t \cdot V$  // $X$ is $\text{Geo}(1 - e^{-1/t})$
13. $Y \leftarrow \lfloor X/s \rfloor$  // $Y$ is $\text{Geo}(1 - e^{-s/t})$
14. return $Y$
```

6.3.3 Secure Implementations of Noisy Top-k with Gap

A common approach to making an algorithm secure on finite machines is to take the ideal algorithm $M$ (that assumes infinite precision), define a rounded version $M'$ where $M'$ rounds the output of $M$, then create a secure algorithm for $M'$. We follow this approach, starting with the ideal Noisy Top-K with Gap algorithm introduced in Chapter 5. For readability we repeat this algorithm below as Algorithm 26.

Algorithm 26: Noisy Top-k with Gap

```plaintext
function GapTopK($q_1, \ldots, q_n, k, \epsilon$):

1. for $i = 1, \ldots, n$ do
2.     $X \leftarrow \text{Exp}(2k/\epsilon)$
3.     $\bar{q}_i \leftarrow q_i + X$
4. $j_1, \ldots, j_{k+1} \leftarrow \arg \max_{k+1}(\bar{q}_1, \ldots, \bar{q}_n)$
5. for $i = 1, \ldots, k$ do
6.     $g_i \leftarrow \bar{q}_{j_i} - \bar{q}_{j_{i+1}}$
7. return $(j_1, g_1), \ldots, (j_k, g_k)$
```

The first step is to create a rounded version of Algorithm 26. Since the indices are integer values, we just need to round each gap down to the nearest multiple of $\gamma \epsilon$. So we have the following Algorithm 27.
Algorithm 27: Noisy Top-k with Gap (Rounded)

1 function GapTopK(q1, ..., qn, k, ε):
2 for i = 1, ..., n do
3 X ← Exp(2k/ε)
4 qi ← qi + X
5 j1, ..., jk+1 ← arg max_{k+1} (qi, ..., q_n)
6 for i = 1, ..., k do
7 gi ← ⌊qi_j - q_{j+1}⌋γ∗
8 return (j1, g1), ..., (jk, gk)

Theorem 6.2. Algorithm 27 is ε-differentially private.

Proof. This is clear because Algorithm 27 is simply Algorithm 26 followed by a post-processing step of rounding every gap down to the closest multiple of γ∗. □

The outputs of Algorithm 27 are floating-point representable. However, the decision variables (e.g., q1, ..., q_n used to determine the selected indices) are not representable because they still involve infinite precision exponential noise. This is an overkill because once the indices are selected, the output gaps are rounded to a target resolution of γ∗. The key idea of the next algorithm is that it tries to find the right resolution of discrete noise to use. If the current resolution is not enough to unambiguously identify the winner and the rounded gap (to a target resolution of γ∗) then it increases the resolution. Using this idea, we have Algorithm 28.

Theorem 6.3. Algorithm 28 is equivalent to Algorithm 27 and therefore is ε-differentially private.

Algorithm 28 differs from Algorithm 27 in that it uses a while-loop to identify the top k + 1 queries. In Algorithm 27, the top k + 1 queries are identified by simply calling the function argmax_{k+1} on q1, ..., q_n in Line 5, which returns the indices of the largest k + 1 queries in descending order. In Algorithm 28, however, we use a while-loop (Line 8) to determine the largest k + 1 noisy queries. In each iteration t, the resolution γ_t (starting with γ∗) is refined by a factor of 1/M. The noisy query answers q_t are rounded down to the nearest multiple of γ_t, denoted by q_t^{(t)}. After that, the argmax_{k+2} function is called on q_{1}^{(t)}, ..., q_{n}^{(t)}, which returns the indices of the top k + 2 values among q_{1}^{(t)}, ..., q_{n}^{(t)} in descending order. Then a for-loop is used to check for ties among the top k + 2 values. When there is no tie among them, the stopping flag is set and the while-loop terminates.
Algorithm 28: Noisy Top-k with Gap (Model)

1 function GapTopK(q1, ..., qn, k, ε):
2     for i = 1, ..., n do
3         X ← Exp(2k/ε)
4         q̂i ← qi + X
5         γ0 ← γ*
6         t ← 1
7         tie ← true
8         while tie do
9             γt ← 1/Mγt-1
10                for i = 1, ..., n do
11                    q̂i(t) ← [q̂i]γt
12                    j1, ..., jk+2 ← arg maxk+2(q̂1(t), ..., q̂n(t))
13                    for i = 1, ..., k + 1 do
14                        if q̂j1(t) == q̂j2(t) then
15                            break
16                            tie ← false
17                            t ← t + 1
18                for i = 1, ..., k do
19                    gi(t) ← [q̂ji - q̂j(i+1)]γt
20         return (j1, g1), ..., (jk, gk)

In Algorithm 28 the decision variables q̂i(t) are floating-point representable because they are multiples of γt. However, they still need to be computed from q̂i(t) which uses continuous exponential noise.

The last step to make our algorithm secure is to compute q̂i(t) using distributions that we can securely sample from. The key observation is the following: just as a finite precision sample from the uniform distribution on [0, 1) can be obtained by repeatedly sampling digits from \( \{0, 1, \ldots, 9\} \), a finite precision sample from the exponential distribution can be obtained by repeatedly sampling from the geometric distribution, which can be done using the secure primitives in Section 6.3.2. Thus we have Algorithm 29.

Theorem 6.4. Algorithm 29 is equivalent to Algorithm 28 and therefore is ε-differentially private.
Algorithm 29: Noisy Top-k with Gap (Secure)

1. function GapTopK(q₁, ..., qₙ, k, e):
2. for i = 1, ..., n do
3.     Y₀ ← Geo(1 - e⁻°(γᵣ/2k))
4.     qᵢ(0) ← qᵢ + γᵣ · Y₀
5.     γ₀ ← γᵣ
6.     t ← 1
7.     tie ← true
8. while tie do
9.     γₜ ← 1/M γₜ-1
10.    for i = 1, ..., n do
11.        Yₜ ← Geo(1 - e⁻°(γᵣ/2k))
12.        qᵢ⁽ᵗ⁾ ← γᵣ · (Yₜ mod M)
13. j₁, jₖ+2 ← arg maxₖ+2(q₁⁽ᵗ⁾, ..., qₙ⁽ᵗ⁾)
14.    for i = 1, ..., k + 1 do
15.        if a_j_i⁽ᵗ⁾ == a_j_i⁽ᵗ⁻¹⁾ then
16.            break
17.        tie ← false
18.        t ← t + 1
19. x₁, ..., xₖ₊₁ ← Shuffle(1, ..., k + 1) // Fisher-Yates shuffle
20.    for i = 1, ..., k do
21.        if xᵢ < xᵢ₊₁ then
22.            gᵢ ← ⌊a_j_i⁽ᵗ⁾ - a_j_i₊₁⁽ᵗ⁾ - γₜ⌋ γᵣ
23.        else
24.            gᵢ ← ⌊a_j_i⁽ᵗ⁾ - a_j_i₊₁⁽ᵗ⁾⌋ γᵣ
25. return (j₁, g₁), ..., (jₖ, gₖ)

6.4 Proofs

6.4.1 Proof of Theorem 6.3

Proof: First, we show that Algorithm 28 terminates with probability 1. Since q₁, ..., qₙ use continuous noises, with probability 1 there is no tie among q₁, ..., qₙ. In other words, let δ = minᵢ≠ⱼ|qᵢ - qⱼ|, then we have δ > 0. Thus when γₜ = γᵣ/M ≤ δ, i.e. t ≥ logₘ(δ⁻¹), we have no tie among q₁⁽ᵗ⁾, ..., qₙ⁽ᵗ⁾. Therefore, with probability 1 the while-loop on Line 8 terminates (in ⌈logₘ(δ⁻¹)⌉ steps).

Next, we show that when the while-loop terminates, the indices j₁, ..., jₖ₊₁ are indeed for the top k + 1 queries. Note that ⌈x⌉ > ⌈y⌉ ∀ x > y. By the termination condition, at the end of
the while-loop we have that \( \tilde{q}_{j_1} > \ldots > \tilde{q}_{j_{k+1}} > \tilde{q}_{s}, s \notin \{j_1, \ldots, j_{k+1}\} \). Therefore, we have \( \tilde{q}_{j_1} > \ldots > \tilde{q}_{j_{k+1}} > \tilde{q}_{s}, s \notin \{j_1, \ldots, j_{k+1}\} \). Thus \( j_1, \ldots, j_{k+1} \) are the indices of the largest \( k + 1 \) queries among \( \tilde{q}_1, \ldots, \tilde{q}_n \). This means if \( \tilde{q}_1, \ldots, \tilde{q}_n \) are the same in Algorithm 27 and Algorithms 28, then the indices \( j_1, \ldots, j_{k+1} \) are the same. Since these two algorithms are identical except for the parts where the top \( k + 1 \) indices are determined, they are equivalent.

\[ \Box \]

### 6.4.2 Proof of Theorem 6.4

To prove the equivalence of Algorithm 28 and Algorithm 29, we need a few lemmas to establish the connection between exponential noise and geometric noise.

**Lemma 6.5.** Let \( X \sim \text{Exp}(\beta) \) and \( Y = \lfloor X \rfloor \gamma \). Then the distribution of \( Y \) is the scaled geometric distribution over \( \{0, \gamma, 2\gamma, \ldots\} \) with success probability \( p = 1 - e^{-\gamma/\beta} \). Alternatively, \( Y \) is \( \gamma \) times a geometric random variable (over \( \{0, 1, 2, \ldots\} \)) with success probability \( p = 1 - e^{-\gamma/\beta} \).

**Proof.** For any value \( m \in \{0, 1, \ldots\} \) we have

\[
P(Y = m\gamma) = P(m\gamma \leq X < (m+1)\gamma) = \int_{m\gamma}^{(m+1)\gamma} \frac{1}{\beta} e^{-\frac{s}{\beta}} ds = e^{-\frac{m\gamma}{\beta}} - e^{-\frac{(m+1)\gamma}{\beta}} = e^{-\frac{m\gamma}{\beta}} (1 - e^{-\frac{\gamma}{\beta}}) = (1 - p)^m p
\]

where we let \( p = 1 - e^{-\gamma/\beta} \) and hence \((1 - p)^m = e^{-m\gamma/\beta}\). \( \Box \)

**Lemma 6.6.** Let \( X \sim \text{Exp}(\beta) \) and \( Y = \lfloor X \rfloor \gamma \). Let \( X' = X - Y \). Then \( X' \) follows the truncated exponential distribution on \( [0, \gamma) \).

**Proof.** Clearly from definition we have \( X' \in [0, \gamma) \). For any \( x \in [0, \gamma) \),

\[
X' \leq x \implies X - \lfloor X \rfloor \gamma \leq x \implies X \in [l\gamma, l\gamma + x], l = 0, 1, \ldots
\]

Thus

\[
P(X' \leq x) = \sum_{l=0}^{\infty} P(l\gamma \leq X < (l+1)\gamma + x) = \sum_{l=0}^{\infty} \int_{l\gamma}^{(l+1)\gamma} \frac{1}{\beta} e^{-\frac{s}{\beta}} ds
\]
For any value $n \gamma$ is already a multiple of $2$.

Let $Y \sim Geo(p)$ and $Y' = Y \mod M$ for some $M > 1$. Then the distribution of $Y'$ is the truncated geometric distribution on $\{0, \ldots, M - 1\}$ with success probability $p$.

Proof. For any value $m \in \{0, \ldots, M - 1\}$

$$P(Y' = m) = \sum_{l=0}^{\infty} P(Y = lM + m) = \sum_{l=0}^{\infty} (1 - p)^{|lM + m|} p = (1 - p)^{m} p \sum_{l=0}^{\infty} (1 - p)^{lM}$$

$$= \frac{(1 - p)^{m} p}{1 - (1 - p)^{M}}$$

Lemma 6.8. Let $X \sim Exp(\beta)$. Let $Y_1 = \lfloor X \rfloor_{\gamma_1}$ and $Y_2 = \lfloor X \rfloor_{\gamma_2}$ with $\gamma_1 = M \gamma_2$ for some $M > 1$. Then $Y_1 = \lfloor Y_2 \rfloor_{\gamma_1}$ and $P(Y_2 \mid Y_1)$ is the same as the probability mass function of $Y' = Y_1 + \gamma_2 (Y \mod M)$ where $Y \sim Geo(1 - e^{-\frac{\gamma_2}{\beta}})$.

Proof. First we show $Y_1 = \lfloor Y_2 \rfloor_{\gamma_1}$, i.e., $[X]_{\gamma_1} = \lfloor [X]_{\gamma_2} \rfloor_{\gamma_1}$. Intuitively, this is clear as rounding to a finer resolution first then to a coarser resolution is equivalent to rounding directly to the coarser resolution.

Let $Y_1 = n \gamma_1$, then $X = n \gamma_1 + s$ for some $s \in [0, \gamma_1)$. Thus we have $Y_2 = \lfloor X \rfloor_{\gamma_2} = n \gamma_1 + \lfloor s \rfloor_{\gamma_2}$ because $n \gamma_1$ is already a multiple of $\gamma_2$. Since $\lfloor s \rfloor_{\gamma_2} < s < \gamma_1$, $[Y_2]_{\gamma_1} = n \gamma_1 = Y_1$. Moreover, for $m \in \{0, \ldots, M - 1\}$:

$$P(Y_2 = n \gamma_1 + m \gamma_2 \mid Y_1 = n \gamma_1)$$

$$= P(n \gamma_1 + m \gamma_2 \leq X < n \gamma_1 + m \gamma_2 + \gamma_2 \mid n \gamma_1 \leq X < n \gamma_1 + \gamma_1)$$

$$= \int_{n \gamma_1}^{n \gamma_1 + m \gamma_2 + \gamma_2} \frac{1}{\beta} e^{-\frac{s}{\beta}} ds = \frac{e^{-\frac{(n \gamma_1 + m \gamma_2)}{\beta}} (1 - e^{-\frac{\gamma_2}{\beta}})}{e^{-\frac{n \gamma_1}{\beta}} (1 - e^{-\frac{\gamma_1}{\beta}})}$$
\[
\frac{(1 - e^{-\gamma_2})}{(1 - e^{-\gamma_1})} \cdot e^{-m\gamma_2} \quad \text{(let } p = 1 - e^{-\gamma_2}) \\
\frac{p(1-p)^m}{1 - (1-p)^m}
\]

Thus from Lemma 6.7, this is the same as \( Y_2 = Y_1 + \gamma_2(Y \mod M) \) where \( Y \) is Geo\((1 - e^{-\gamma_2})\). □

**Lemma 6.9.** Let \( Y_1, Y_2 \) be independent random variables of geometric distribution with success probabilities \( p_1 \) and \( p_2 \) such that \( 1 - p_1 = (1 - p_2)^M \) for some \( M > 1 \). Let \( \gamma_1 = M\gamma_2 \) and \( Y = \gamma_1 Y_1 + \gamma_2(Y_2 \mod M) \). Then \( P(Y) \) is the scaled geometric distribution over \( \{0, \gamma_2, 2\gamma_2, \ldots\} \) with success probability \( p_2 \).

**Proof.** Clearly, the range of \( Y \) is \( \{0, \gamma_2, 2\gamma_2, \ldots\} \). For any value \( m\gamma_2 \in \{0, \gamma_2, 2\gamma_2, \ldots\} \), write it uniquely in the form \( m\gamma_2 = k\gamma_1 + n\gamma_2 \) with \( 0 \leq n \leq M-1 \). Then we have \( m = kM + n \) and

\[
P(Y = m\gamma_2) = P(Y_1 = k, (Y_2 \mod M) = n) = P(Y_1 = k) \cdot P((Y_2 \mod M) = n) \\
= (1 - p_1)^k \cdot p_1 \cdot \frac{(1 - p_2)^n \cdot p_2}{1 - (1 - p_2)^M} \\
= (1 - p_2)^{kM} \cdot (1 - (1 - p_2)^M) \cdot \frac{(1 - p_2)^n \cdot p_2}{1 - (1 - p_2)^M} \\
= (1 - p_2)^{kM+n} \cdot p_2 = (1 - p_2)^m \cdot p_2
\]

The second line is because \((Y_2 \mod M)\) follows the truncated geometric distribution from Lemma 6.7. □

Now we are ready to establish the equivalence between Algorithm 28 and Algorithm 29.

**Proof of Theorem 6.4.** The outputs of both algorithms consist of two sets of values: the indices \( j_1, \ldots, j_k \) which identify the top \( k \) queries in descending order, and \( g_1, \ldots, g_k \) which are the numeric gaps between each of the identified query and its runner-up. We first show that the indices from Algorithm 28 follow the same distribution as those from Algorithm 29. Note that \( j_1, \ldots, j_k \) are determined by \( \tilde{q}_i^{(r)} \) in Algorithm 28 and by \( \tilde{q}_i^{(r)} \) in Algorithm 29 respectively. Therefore, it suffices to show that for all \( T \geq 1 \), the set of random variables \( (\tilde{q}_1^{(1)}, \ldots, \tilde{q}_n^{(1)}, \ldots, \tilde{q}_1^{(T)}, \ldots, \tilde{q}_n^{(T)}) \) in Algorithm 28 and \( (\tilde{q}_1^{(1)}, \ldots, \tilde{q}_n^{(1)}, \ldots, \tilde{q}_1^{(T)}, \ldots, \tilde{q}_n^{(T)}) \) in Algorithm 29 have the same distribution. Also note that for
$i_1 \neq i_2$, the set of variables $\{\tilde{q}_{i_1}^{(1)}, \ldots, \tilde{q}_{i_2}^{(T)}\}$ is independent of the set $\{\tilde{q}_{i_2}^{(1)}, \ldots, \tilde{q}_{i_2}^{(T)}\}$, thus

$$P(\tilde{q}_1^{(1)}, \ldots, \tilde{q}_n^{(1)}, \ldots, \tilde{q}_1^{(T)}, \ldots, \tilde{q}_n^{(T)}) = \prod_{i=1}^n P(\tilde{q}_i^{(1)}, \ldots, \tilde{q}_i^{(T)}).$$

Similarly, for Algorithm 29 we have

$$P(\tilde{q}_1^{(1)}, \ldots, \tilde{q}_n^{(1)}, \ldots, \tilde{q}_1^{(T)}, \ldots, \tilde{q}_n^{(T)}) = \prod_{i=1}^n P(\tilde{q}_i^{(1)}, \ldots, \tilde{q}_i^{(T)}).$$

Thus it suffices to show that $\forall i \in \{1, \ldots, n\}$ and $\forall T \geq 1$,

$$P(\tilde{q}_i^{(1)}, \ldots, \tilde{q}_i^{(T)}) = P(\tilde{q}_i^{(1)}, \ldots, \tilde{q}_i^{(T)}).$$

Recall that in Algorithm 28, $\tilde{q}_i = q_i + X$ where $X \sim \text{Exp}(2k/e)$ is random noise from the exponential distribution. Furthermore, in the $t^{th}$ iteration of the loop we have $\tilde{q}_i^{(t)} = [\tilde{q}_i]_{\gamma_t} = q_i + [X]_{\gamma_t}$ since $q_i$ is already a multiple of $\gamma_*$ and hence also a multiple of $\gamma_t = \gamma_* / M'$. On the other hand, in Algorithm 29 we have $\tilde{q}_i^{(t)} = q_i + \gamma_0 Y_0 + \ldots + \gamma_t Y_t$ in the $t^{th}$ iteration of the loop. Let $V_t = [X]_{\gamma_t}$ and $W_t = \sum_{j=0}^t \gamma_j Y_j$. Then we just need to show that $P(V_1, \ldots, V_T) = P(W_1, \ldots, W_T)$. We do this by induction on $T$.

For $T = 1$, $V_1 = [X]_{\gamma_1}$ follows the $\gamma_1$-scaled geometric distribution with success probability $p = 1 - e^{-\frac{\epsilon \gamma_1}{M}}$ from Lemma 6.5. Also, we have $W_1 = \gamma_0 Y_0 + \gamma_1 Y_1$ where $Y_0$ and $Y_1$ are noises from the geometric distribution with success probabilities $p' = 1 - e^{-\frac{\epsilon \gamma_0}{M}}$ and $p'' = 1 - e^{-\frac{\epsilon \gamma_1}{M}}$. Since $\gamma_0 = M \gamma_1$, we have $1 - p' = e^{-\frac{\gamma_0}{M}} = (e^{-\frac{\gamma_1}{M}})^M = (1 - p'')^M$. Therefore, by Lemma 6.9 $W_1$ also follows the $\gamma_1$-scaled geometric distribution with success probability $p'' = p$. So $P(V_1) = P(W_1)$.

Now assume as the inductive hypothesis that we have $P(V_1, \ldots, V_T) = P(W_1, \ldots, W_T)$ for some $T \geq 1$. First, note that $P(V_1, \ldots, V_{T+1}) = P(V_{T+1} \mid V_1, \ldots, V_T)P(V_1, \ldots, V_T)$. Furthermore, by Lemma 6.8 we have $V_t = [V_T]_{\gamma_t}$ for $t = 1, \ldots, T - 1$. Thus the values of $V_1, \ldots, V_{T-1}$ are all determined by $V_T$ and hence $P(V_{T+1} \mid V_1, \ldots, V_T) = P(V_{T+1} \mid V_T)$. Therefore, from Lemma 6.8 again we have that $V_{T+1} = V_T + \gamma_{T+1}(Y \mod M)$ where $Y \sim \text{Geo}(1 - e^{-\frac{\epsilon \gamma_{T+1}}{M}})$.

On the other hand, $P(W_1, \ldots, W_{T+1}) = P(W_{T+1} \mid W_T)P(W_1, \ldots, W_T)$. By definition we have $W_{T+1} = W_T + \gamma_{T+1}(Y_{T+1} \mod M)$ with $Y_{T+1} \sim \text{Geo}(1 - e^{-\frac{\epsilon \gamma_{T+1}}{M}})$. Therefore, we have $P(V_{T+1} \mid V_T) = P(W_{T+1} \mid W_T)$ and thus $P(V_1, \ldots, V_{T+1}) = P(W_1, \ldots, W_{T+1})$. This completes the inductive
step. Therefore, we have \( P(V_1, \ldots, V_T) = P(W_1, \ldots, W_T), \forall T \). Hence the output indices from both Algorithms follow the same distribution.

Lastly we show that the gaps \( g_1, \ldots, g_k \) from the two algorithms follow the same distribution. Note that \( [X_i - X_{i+1}]_\gamma = ([X_i - X_{i+1}]_\gamma, \gamma), \) we just need to show that \( [X_i - X_{i+1}]_\gamma \) follows the same distribution as \( W_i - W_{i+1} - \delta_i \gamma \), where \( \delta_i \in \{0, 1\} \) is determined by the for-loop in Algorithm 29 from Line 20 to Line 24. This follows from the next lemma.

**Lemma 6.10.** Let \( X_1, \ldots, X_{k+1} \) \( \overset{i.i.d.}{\sim} \) \( \text{Exp}(\beta) \). Let \( Y_i = [X_i]_\gamma \) and \( G_i = [X_i - X_{i+1}]_\gamma \). Then \( G_i \) is either \( Y_i - Y_{i+1} \) or \( Y_i - Y_{i+1} - \gamma \), and

\[
P(G_i = Y_i - Y_{i+1} - \delta_i \gamma, i \in [k] \mid Y_i, i \in [k+1]) = P((-1)^{\delta_i} (\pi(i) - \pi(i + 1)) > 0, i \in [k])
\]

where \( \pi \) is a random permutation of \( \{1, \ldots, k+1\} \) and \( \delta_i \in \{0, 1\} \).

**Proof.** From the definition \( Y_i = [X_i]_\gamma \) we have that \( X_i = Y_i + X'_i \) where \( X'_i \in [0, \gamma) \). Therefore, \( X'_i - X'_{i+1} \in (-\gamma, \gamma) \) and we have

\[
G_i = [X_i - X_{i+1}]_\gamma = Y_i - Y_{i+1} + [X'_i - X'_{i+1}]_\gamma = \begin{cases} Y_i - Y_{i+1} - \gamma & X'_i - X'_{i+1} \in (-\gamma, 0) \\ Y_i - Y_{i+1} & X'_i - X'_{i+1} \in [0, \gamma) \end{cases}
\]

From Lemma 6.6 we have \( X'_i \overset{i.i.d.}{\sim} \text{Exp}(\beta) \mod \gamma \). Thus for every permutation \( \pi \) on \( [k+1] \) it is equally likely that \( X'_{\pi(1)} < X'_{\pi(2)} < \ldots < X'_{\pi(k+1)} \). Hence we can use a random permutation \( \pi \) to determine the relative order of \( X'_i \): \( X'_{\pi(i)} < X'_{\pi(j)} \iff i < j \), or equivalently, \( X'_i < X'_j \iff \pi^{-1}(i) < \pi^{-1}(j) \).

Since \( \pi^{-1} \) is also a random permutation, we will drop the inverse and just use \( \pi \). Therefore,

\[
P(G_i = Y_i - Y_{i+1} - \delta_i \gamma, i \in [k] \mid Y_i, i \in [k+1]) = P((-1)^{\delta_i} (X'_i - X'_{i+1}) > 0, i \in [k])
\]

\[= P((-1)^{\delta_i} (\pi(i) - \pi(i + 1)) > 0, i \in [k]).
\]

where \( \pi \) is a random permutation on \( [k+1] \). \( \square \)
Chapter 7

Conclusion

While it is invaluable to formally verify correct differentially-private algorithms, we believe that it is equally important to detect incorrect algorithms and provide counterexamples for them, due to the subtleties involved in algorithm development. In this dissertation, we proposed a novel semi-black-box method of evaluating differentially private algorithms, and providing counterexamples for those incorrect ones. We show that within a few seconds, our tool correctly rejects incorrect algorithms (including published ones) and provides counterexamples for them.

In this dissertation we also introduced variations of private selection algorithms including Sparse Vector Technique, Noisy Top-$k$ and Exponential Mechanism that provide additional noisy gap information for free (without affecting the privacy cost). We presented applications of how to use the gap information. Evaluations on a variety of datasets show that this extra information can significantly increase the accuracy of differentially private query estimates. Lastly, we presented a secure implementation of our improved Noisy Top-$k$ algorithm that is immune to attacks that exploit the floating-point vulnerability and proved its correctness.
Chapter 8

Future Work

In this chapter, we sketch a few interesting future directions.

**Generalizing the Proof Template** Our current proof template and verification tools rely on the underlying randomness alignment technique; hence they are subject to its limitations, including lack of support for approximate \((\epsilon, \delta)\)-differential privacy [28], (zero) concentrated differential privacy [29, 20] and Rényi differential privacy [21]. We are currently working to extend the underlying proof technique for other variants of differential privacy and to support more noise distributions such as Gaussian.

**Private Database Engines** A SQL engine that can answer aggregate queries with differential privacy is vitally important for organizations that need to share analytics about sensitive datasets. To be suitable for production, such a system needs to support flexible privacy modeling and have the capabilities to optimize queries for accuracy and verify privacy guarantees. In a separate on-going project, we are designing a differential privacy layer and a domain specific language that is compatible with existing SQL databases and provides the aforementioned desiderata.

**Private Data Collection and Release** The traditional model for differential privacy assumes that there is a trusted curator who has direct access to the data. In practice, however, the first large deployments of differential privacy (e.g. Google’s RAPPOR [2]) operated in the distributed (local) model in which there is no trusted curator, and random perturbation is applied on device. The local model provides stronger privacy protections for individuals as they do not need to trust the data collector. It has enabled companies to collect potentially sensitive data from clients in a privacy-preserving manner with strong local differential privacy (LDP) guarantees. LDP protocols can help the data collector estimate the frequency of any value, or find the most frequent values, also known as the heavy hitters, among all users. This could allow companies, for example, to estimate the percentage of users who searched a
specific keyword, or the most searched keywords on the internet. Current technologies, however, do not scale well for very large and unknown domains. In collaboration with industry, we are developing new protocols that work with various forms of data such as key-value pairs and audio at scale.

Another interesting area of research is private data release. Social media companies such as Facebook have enormous data that could be leveraged to study some of the most important questions of the day—including the spread of misinformation and the dynamics of election interference. In January 2020, Facebook released the Condor URLs Dataset [111] which allowed outside researchers to study the distribution of URLs on Facebook and how users interacted with them. In close collaboration with industry, I plan to explore differentially private natural language generation (NLG) techniques to generate synthetic textual datasets that resemble the actual messages posted on social media to facilitate research on misinformation from across the web, shared and spread on social media platforms.

**Machine Learning with Private Data**  It is well known in the literature that machine learning models could leak sensitive information. In the field of natural language processing, large language models, which underpins many of the most powerful neural networks, could memorize the full name, physical address, email address and phone number of an individual [112]. Thus it is vitally important to ensure that the sensitive information in the training data is protected when the trained model is released to the public. Textual data on social media often come with additional information, e.g., the user who owns the text and other auxiliary features. Leveraging this extra information, we are building a framework for fine-tuning large language models for specific downstream tasks on private datasets with user level privacy. We plan to investigate novel dimension reduction methods to reduce the number of parameters to be updated and privacy accounting techniques to achieve tight and efficient privacy computation.

**Privacy and Security in Emerging Technologies**  Looking further out, I plan to investigate the privacy and security concerns in several emerging technologies, including Augmented Reality (AR) and Virtual Reality (VR). AR/VR systems are essentially collections of sensors and displays that work in concert to create an immersive experience for the user. In order for them to function, these systems collect a lot of highly personal information about who the user is and what they are doing— to a much greater extent than, for example, social media networks. Moreover, advanced technologies, such as eye-tracking and brain-computer interface (BCI) technologies that interpret neural signals, collect extensive biometric
data, which can identify individuals and infer additional information. Emerging technologies raise new user privacy considerations, and research in these fields are in an early and immature stage. Now is a good time to systematically study the privacy risks in these systems and pioneer defense strategies.
Bibliography


URL http://doi.acm.org/10.1145/3132747.3132769


URL http://dl.acm.org/citation.cfm?id=3294996.3295115


URL http://doi.acm.org/10.1145/3035918.3035940


Vita
Ding Ding

EDUCATION

Pennsylvania State University
Ph.D. in Computer Science and Engineering
State College, PA
August 2022 (expected)

SUNY Binghamton
Ph.D. in Mathematics
Binghamton, NY
December 2015

Zhejiang University
B.S. in Mathematics
Hangzhou, China
June 2006

EXPERIENCE

Facebook Inc.
Research Intern
Menlo Park, CA
May 2021 - December 2021

Baidu Inc.
Research Intern
Sunnyvale, CA
May 2020 - August 2020

HONORS & AWARDS

• Best Paper Award, ACM Conference on Computer and Communications Security, 2018

• Graduate Research Award, Penn State University, 2019

• The Caspar Bowden PET Award Runner-Up, 2019

• Best Paper Award Runner-Up, ACM Conference on Computer and Communications Security, 2020

• Graduate Teaching Award, Penn State University, 2021

• Best Paper Award Runner-Up, ACM Conference on Computer and Communications Security, 2021

SERVICE

Program Committee:
• ACM Conference on Computer and Communications Security (CCS), 2022

Reviewer:
• The 39th International Conference on Machine Learning (ICML), 2022
• The International Conference on Learning Representations (ICLR), 2022
• Conference on Neural Information Processing Systems (NeurIPS), 2021
• ACM/IEEE Symposium on Logic in Computer Science (LICS), 2021
• ACM Conference on Computer and Communications Security (CCS), 2019