Widespread Underestimation of Sensitivity in Differentially Private Libraries and How to Fix It

Sílvia Casacuberta
Harvard University
Cambridge, MA, USA
scasacubertapuig@college.harvard.edu

Michael Shoemate
Harvard University
Cambridge, MA, USA
shoematem@seas.harvard.edu

Salil Vadhan
Harvard University
Cambridge, MA, USA
salil_vadhan@harvard.edu

Connor Wagaman
Boston University
Boston, MA, USA
wagaman@bu.edu

ABSTRACT

We identify a new class of vulnerabilities in implementations of differential privacy. Specifically, they arise when computing basic statistics such as sums, thanks to discrepancies between the implemented arithmetic using finite data types (namely, ints or floats) and idealized arithmetic over the reals or integers. These discrepancies cause the sensitivity of the implemented statistics (i.e., how much one individual's data can affect the result) to be much larger than the sensitivity we expect. Consequently, essentially all differential privacy libraries fail to introduce enough noise to hide individual-level information as required by differential privacy, and we show that this may be exploited in realistic attacks on differentially private query systems. In addition to presenting these vulnerabilities, we also provide a number of solutions, which modify or constrain the way in which the sum is implemented in order to recover the idealized or near-idealized bounds on sensitivity.

CCS CONCEPTS

• **Security and privacy → Information-theoretic techniques**;
• **Mathematics of computing → Statistical software**.

KEYWORDS

differential privacy; finite-precision arithmetic; floating-point numbers; privacy attacks

ACM Reference Format:
Sílvia Casacuberta, Michael Shoemate, Salil Vadhan, and Connor Wagaman. 2022. Widespread Underestimation of Sensitivity in Differentially Private Libraries and How to Fix It. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS'22).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560708

CCS'22, November 7–11, 2022, Los Angeles, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560708

1 INTRODUCTION

Differential privacy (DP) [14] has become the prevailing framework for protecting individual-level privacy when releasing statistics or training machine learning models on sensitive datasets. It has been the subject of a rich academic literature across many areas of research, and has seen major deployments by the US Census Bureau [34, 17, 2], Google [16, 4, 5, 6], Apple [44], Facebook/Meta [36, 23], Microsoft [11, 40], LinkedIn [41], and OhmConnect.[1] To facilitate the adoption of differential privacy, a number of researchers and organizations have released open-source software tools for differential privacy, starting with McSherry's PINQ [35], and now including libraries and systems from companies like Google [21], Uber [29], IBM [26], and Facebook/Meta [48], and the open-source projects OpenMined[2] and OpenDP [18].

However, implementing differential privacy correctly is subtle and challenging. Previous works have identified and attempted to address vulnerabilities in implementations of differential privacy coming from side channels such as timing [22, 28] and the failure to faithfully emulate the noise infusion mechanisms needed for privacy when using floating-point arithmetic instead of idealized real arithmetic [37, 27, 28].

In this work, we identify a new and arguably more basic class of vulnerabilities in implementations of differential privacy. Specifically, they arise when computing basic statistics such as sums, thanks to discrepancies between the implemented arithmetic using finite data types (namely, ints or floats) and idealized arithmetic over the reals or integers. These discrepancies cause the *sensitivity* of the implemented statistics — how much one individual's data can affect the result — to be much larger than the sensitivity we expect. Consequently, essentially all differential privacy libraries fail to introduce enough noise to hide individual-level information as required by differential privacy, and we show that this may be exploited in realistic attacks on differentially private query systems. In addition to presenting these vulnerabilities, we also provide a number of solutions, which modify or constrain the way in which the sum is implemented in order to recover the idealized or near-idealized bounds on sensitivity.

In this paper, we give an overview of our results. Precise definitions, proofs, and more details about the attack experiments are

[1] https://edp.recurve.com/.
[2] https://github.com/OpenMined/PyDP.

provided in the full version of the paper, which can be found at https://arxiv.org/abs/2207.10635 [10].

1.1 Differential Privacy

Informally, a randomized algorithm $\mathcal{M}$ is *differentially private* if for every two datasets u, u' that differ on one individual's data, the probability distributions $\mathcal{M}(u)$ and $\mathcal{M}(u')$ are close to each other. Intuitively, this means that an adversary observing the output cannot learn much about any individual, since what the adversary sees is essentially the same as if that individual's data were not used.

To make the definition of differential privacy precise, we need to specify both what it means for two datasets u and u' to "differ on one individual's data," and how we measure the closeness of probability distributions $\mathcal{M}(u)$ and $\mathcal{M}(u')$. For the former, there are two common choices in the differential privacy literature. In one choice, we allow u' to differ from u by adding or removing any one record; this is called *unbounded differential privacy* and thus we denote this relation $u \simeq_{unbdd} u'$. Alternatively, we can allow u' to differ from u by *changing* any one record; this is called *bounded differential privacy* and thus we will write $u \simeq_{bdd} u'$. Note that if $u \simeq_{bdd} u'$, then u and u' necessarily have the same number of records; thus, this is an appropriate definition when the size n of the dataset is known and public. When $u \simeq u'$ for whichever relation we are using ($\simeq_{bdd}$ or $\simeq_{unbdd}$), we call u and u' *adjacent* with respect to $\simeq$.

For measuring the closeness of the probability distributions $\mathcal{M}(u)$ and $\mathcal{M}(u')$, we use the standard definition of (ε, δ)-differential privacy [13], requiring that:

$$\forall T \qquad \Pr[\mathcal{M}(u) \in T] \leq e^{\varepsilon} \cdot \Pr\left[\mathcal{M}(u') \in T\right] + \delta, \tag{1}$$

where we quantify over all sets T of possible outputs. There are now a variety of other choices, like Concentrated DP [15, 8], Rényi DP [38], and f-DP [12]; our results are equally relevant to these forms of DP, but we stick with the basic (ε, δ) notion for simplicity. If $\mathcal{M}$ satisfies (1) for all u, u' such that $u \simeq u'$, then we say that $\mathcal{M}$ is *(ε, δ)-DP with respect to $\simeq$*. If $\delta = 0$, we say $\mathcal{M}$ is *ε-DP with respect to $\simeq$*. Intuitively, ε measures *privacy loss* of the mechanism $\mathcal{M}$, whereas δ bounds the probability of failing to limit privacy loss to ε (so we typically take δ to be cryptographically small).

The fundamental building block of most differentially private algorithms is noise addition calibrated to the *sensitivity* of a function f that we wish to estimate.

Definition 1.1 (Sensitivity)**.** Let f be a real-valued function on datasets, and $\simeq$ a relation on datasets. The *(global) sensitivity* of f with respect to $\simeq$ is defined to be:

$$\Delta_{\simeq} f = \sup_{u \simeq u'} |f(u) - f(u')|.$$

Theorem 1.2 (Laplace Mechanism [14]). *For every function f and relation $\simeq$ on datasets, the mechanism*

$$\mathcal{M}(u) = f(u) + \mathrm{Lap}\left(\frac{\Delta_{\simeq} f}{\varepsilon}\right)$$

is ε-DP, where $\mathrm{Lap}(s)$ denotes a draw from a Laplace random variable with scale parameter s.

There are a number of other choices for the noise distribution, leading to the Discrete Laplace (a.k.a., Geometric) Mechanism [20], the Gaussian Mechanism [39], and the Discrete Gaussian Mechanism [9], where the latter two achieve (ε, δ)-DP with $\delta > 0$. The key point for us is that in all cases, the scale or standard deviation of the noise is supposed to grow linearly with the sensitivity $\Delta_{\simeq} f$, so it is crucial that the sensitivity is calculated correctly. If we incorrectly underestimate the sensitivity as $\Delta = (\Delta_{\simeq} f)/c$ for a large constant c, then we will only achieve $c\varepsilon$-DP; that is, our privacy loss will be much larger than expected. Given that it is common to use privacy loss parameters like $\varepsilon = 1$ or $\varepsilon = 0.5$, a factor of 5 or 10 increase in the privacy-loss parameter can have dramatic effects on the privacy protection (since $e^5 > 148$, allowing a huge difference between the probability distributions $\mathcal{M}(u)$ and $\mathcal{M}(u')$).

The most widely used function f in DP noise addition mechanisms is the *Bounded Sum* function.

Definition 1.3 (Bounded Sum)**.** For real numbers $L \leq U$, and a dataset v consisting of elements of the interval $[L, U]$, we define the *Bounded Sum* function on v to be:

$$BS_{L,U}(v) = \sum_{i=1}^{\mathrm{len}(v)} v_i.$$

The restriction of $BS_{L,U}$ to datasets v of length n is denoted $BS_{L,U,n}$. When we do not constrain the data to lie in $[L, U]$, we omit the subscripts.

Typically, the bounds L and U are enforced on the dataset v via a record-by-record clamping operation. To avoid the extra notation of the clamp operation, throughout we will work with datasets that are assumed to already lie within the bounds.

It is well-known and straightforward to calculate the sensitivity of Bounded Sum.

Proposition 1.4. *$BS_{L,U}$ has sensitivity $\max\{|L|, |U|\}$ with respect to $\simeq_{unbdd}$, and $BS_{L,U,n}$ has sensitivity $U - L$ with respect to $\simeq_{bdd}$.*

Combining Proposition 1.4 and Theorem 1.2 (or analogs for other noise distributions), we obtain a differentially private algorithm for approximating bounded sums, which we will refer to as *Noisy Bounded Sum*. This algorithm is pervasive throughout both the differential privacy literature and software. Many more complex statistical analyses can be decomposed into bounded sums, and any machine learning algorithm that can be described in Kearns' Statistical Query model [32] can be made DP using Noisy Bounded Sum. Indeed, the versatility of noisy sums was the basis of the SuLQ privacy framework [7], which was a precursor to the formal definition of differential privacy. Noisy Bounded Sum is also at the heart of differentially private deep learning [1], as each iteration of (stochastic) gradient descent amounts to approximating a sum (or average) of gradients. For this reason, every software package for differential privacy that we are aware of supports computing noisy bounded sums via noise addition.

1.2 Previous Research

Despite their simplicity, Noisy Bounded Sum and related differentially private algorithms are surprisingly difficult to implement in a way that maintains the desired privacy guarantees.

In the early days of implementing differential privacy, several challenges were pointed out by Haeberlen, Pierce, and Narayan [22]. Their attacks concern not the Bounded Sum function itself, but dataset transformations that are applied before the Bounded Sum function. In a typical usage, Bounded Sum is not directly applied to a dataset $u = [u_1, \ldots, u_n]$, but rather to the dataset

$$q(u) \stackrel{\text{def}}{=} [q(u_1), q(u_2), \ldots, q(u_n)]$$

where q is a "microquery" mapping records to the interval $[L, U]$. (We can incorporate the clamping into q.) If $u \simeq u'$, then $q(u) \simeq q(u')$, so we if we apply Noisy Bounded Sum (or any other differentially private algorithm) to the transformed dataset, we should still satisfy differential privacy with respect to the original dataset u. Haeberlen et al.'s attacks rely on a discrepancy between this mathematical abstraction and implementations. In code, q may not be a pure function, and may be able to access global state (allowing information to flow from the execution of $q(u_i)$ to the execution of $q(u_j)$ for $j > i$) or leak information to the analyst through side channels (such as timing). The authors of the main differentially private systems at the time, PINQ [35] and Airavat [42], were aware of and noted the possibility of such attacks, but the implemented software prototypes did not fully protect against them. As discussed in [35, 42, 22], remedies for attacks like these include using a domain-specific language for the microquery q to ensure that it is a pure function and enforcing constant-time execution for $q(u)$.

At the other end of the DP pipeline (after the calculation of Bounded Sum), the seminal work of Mironov [37] demonstrated vulnerabilities coming from the noise addition step. Specifically, the Laplace distribution in Theorem 1.2 is a continuous distribution on the real numbers, but computers cannot manipulate arbitrary real numbers. Typical implementations approximate real numbers using finite-precision floating-point numbers, and Mironov shows that these approximations can lead to complete failure of the differential privacy property. To remedy this, Mironov proposed the Snapping Mechanism, which adds a sufficiently coarse rounding and clamping after the Laplace Mechanism to recover differential privacy with a slightly larger value of ε. Subsequent works [19, 3, 9] avoided floating-point arithmetic, advocating for and studying the use of exact finite-precision arithmetic (e.g., using big integers) and using discrete noise distributions, such as the discrete Laplace distribution [20] and the discrete Gaussian distribution [9]. However, a recent paper by Jin et al. [28] shows that current implementations of the discrete distribution samplers are vulnerable to timing attacks (which leak information about the generated noise value and hence of the function value it was meant to obscure). They, as well as [25], also show that the floating-point implementations of the continuous Gaussian Mechanism are vulnerable to similar attacks as those shown by Mironov [37] for the Laplace Mechanism.

Ilvento [27] studies the effect of floating-point approximations on another important DP building block, the Exponential Mechanism. On a dataset u, the exponential mechanism samples an outcome y from a finite set $\mathcal{Y}$ of choices with probability $p_y(u)$ proportional to $\exp\left(\varepsilon f_y(u)/(2 \max_y \Delta_{\simeq} f_y)\right)$, where $f_y(u)$ is an arbitrary measure of the "quality" of outcome y for dataset u. She shows that the discrepancy between floating-point and real arithmetic can lead to incorrectly converting the quality scores $f_y(u)$ into the probabilities $p_y(u)$ and hence violate differential privacy. To remedy this, Ilvento proposes an exact implementation of the Exponential Mechanism using finite-precision base 2 arithmetic. Note that, like noise addition mechanisms, the Exponential Mechanism also is calibrated to the sensitivities $\Delta_{\simeq} f_y$ of the quality functions. Here too, if we underestimate the sensitivity by a factor of c, our privacy loss can be greater than intended by a factor of c, even if we perfectly implement the sampling or noise generation step.

Indeed, Mironov's paper [37] also suggests that sensitivity calculations may fail to translate from idealized, real-number arithmetic to implemented, floating-point arithmetic. He gives an example of two datasets $u \simeq_{bdd} u'$ of 64-bit floating-point numbers such that $|BS(u) - BS(u')| = 1$ but $|BS^*(u) - BS^*(u')| = 129$, where BS^* is the standard, iterative implementation of summation of floating-point numbers, illustrated in Figure 1.

```
def iterated_sum(u):
    the_sum = 0
    for element in u:
        the_sum += element
    return the_sum
```

Figure 1: Iterated Summation.

However, Mironov's example does not immediately lead to an underestimation of sensitivity, because the datasets u and u' include values ranging from $L = -2^{-23}$ to $U = 2^{30}$, so the idealized sensitivity suggested by Proposition 1.4 is $U - L > 2^{30} \gg 129$.[3] Mironov's paper suggests that these potential sensitivity issues may be addressed by replacing Iterated Summation by a *Kahan Summation* [31], a different way of summing floating-point numbers that accumulates rounding errors more slowly. Unfortunately, this section of Mironov's paper seems to have gone mostly unnoticed, and we are not aware of any prior work that has addressed the question of how to correctly bound and control sensitivity in finite-precision implementations of differential privacy.

1.3 Our Contributions

In our work, we identify new privacy vulnerabilities arising from underestimation of the sensitivity of the Bounded Sum function when implemented using finite data types, including 32-bit and 64-bit integers and floats. Specifically, implementations of Bounded Sum often have sensitivities much larger than the idealized sensitivity given by Proposition 1.4, and consequently the privacy loss of DP mechanisms using Bounded Sum is much larger than specified. Thus, our work covers vulnerabilities arising from the "middle step" of the DP pipeline — aggregation — sitting between the foci of previous work, which considered vulnerabilities in the preprocessing

[3] As pointed out in Mironov's paper, this example does demonstrate an underestimation of sensitivity by a factor of 129 if we define $u \simeq u'$ to mean that u and u' agree on all but one coordinate and they differ by at most 1 on that coordinate. This notion of dataset adjacency is common in the DP literature when u and u' represent datasets in *histogram* format, where u_i is the number of individuals of type i (rather than individual i's data). However, in this case, the u_i's would be nonnegative integers (so this example would not be possible) and it would be strange to use a floating-point data type.

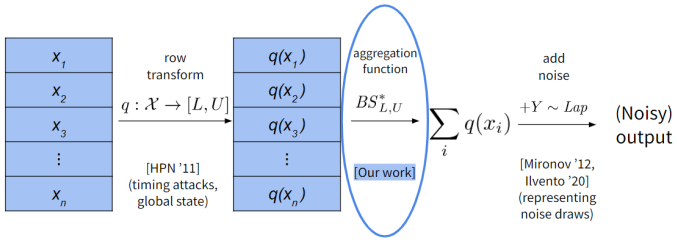

Figure 2: Illustration of the DP pipeline and the relationship between this paper and previous works uncovering vulnerabilities of DP implementations.

step before aggregation and the noise generation/sampling step after aggregation (see Figure 2).

In addition to describing the vulnerabilities that emerge due to sensitivity underestimation in essentially all libraries of DP functions and showing how to exploit them, we describe several solutions that recover the idealized or near-idealized bounds on sensitivity. Many of our solutions only require small modifications to current code.

1.4 Finite-precision Arithmetic in DP Libraries

Before describing our results in more detail, we summarize how existing implementations of differential privacy address arithmetic issues (at the time of our work, prior to fixes implemented in response to our paper). A more complete description of these libraries can be found in the full version of the paper [10]. All current DP implementations make use of the finite-precision data types (e.g., 32-bit or 64-bit ints or floats) to which our attacks apply. Some of the libraries attempt to address the vulnerabilities uncovered by Mironov at the noise addition step [37]. For example, Google's DP library [47] includes sampling algorithms for floating-point approximations to the Laplace and Gaussian distributions (based on Mironov's Snapping Mechanism) which they claim circumvent problems with naïve floating-point implementations.[4] IBM's `diffprivlib` [26] samples a floating-point approximation to the Laplace distribution using the method described in Holohan and Braghin [25]. The OpenDP Library acknowledges[5] the floating point vulnerabilities discovered by Mironov, and users have access to floating-point mechanisms only if the `'contrib'` compilation flag is turned on to allow components that do not have verified proofs.

Indeed, our work can be seen as following the call of the OpenDP Programming Framework paper [18], which says that "any deviations from standard arithmetic (e.g., overflow of floating-point arithmetic) should be explicitly modelled in the privacy analysis." (The paper [18] goes further and advocates the use of fixed-point and integer arithmetic as much as possible. We believe that abandoning floating-point arithmetic entirely may have a significant usability cost, so we consider both solutions that operate only on floating-point numbers and solutions that reduce floating-point summation to integer summation.)

[4]https://github.com/google/differential-privacy/blob/main/common_docs/Secure_Noise_Generation.pdf.
[5]https://docs.opendp.org/en/stable/user/measurement-constructors.html#floating-point

In other DP software (e.g., Opacus [1], Chorus [30], Airavat [42], PINQ [35]), we did not find any mention of potential issues or solutions for floating-point computations.

All of the libraries we studied scale noise according to the idealized sensitivity of the Bounded Sum function (Proposition 1.4):

- Unbounded DP: Google's sum function, SmartNoise's sum function, Opacus, and Airavat [42, §4.1].
- Bounded DP: IBM `diffprivlib`'s sum and mean, Google's mean, Chorus [30, §3] and SmartNoise's sized sum function.

All of these libraries underestimate the sensitivity of the implemented Bounded Sum function, for both integer and floating-point data types, and thus are vulnerable to our attacks. We remark that carrying out our attacks in practice (to extract sensitive individual information from real-life datasets) does seem to require an adversary to carefully choose a microquery/row-transform q (see Figure 2), so these vulnerabilities are more of an immediate threat when the DP software is used as part of an interactive query system rather than for noninteractive data releases. However, the fact that the DP guarantee fails raises the possibility of other attacks, which may not require interactive queries; this possibility can be avoided by implementing one of our solutions to recover a correct proof of differential privacy.

1.5 Organization

In Section 2, we present the necessary notation for the paper. We present an overview of the attacks that yield the sensitivity lower bounds for both bounded and unbounded DP in Section 3. We identify four different types of vulnerabilities: overflow, rounding, repeated rounding, and reordering. In Section 4 we show how these attacks can be carried out on the main existing DP libraries. Lastly, in Section 5 we summarize the different solutions that we propose to fix these vulnerabilities: dataset adjacency relations, random permutations, checking or bounding parameters, truncated summation, split summation, sensitivity from accuracy, shifting bounds, and reducing floats to ints. In Section 6 we propose a roadmap for DP libraries with recommendations on how to best prioritize and implement our solutions.

The full version of the paper [10] contains a more complete presentation of preliminaries; formal theorem statements, proofs, and further examples of vulnerabilities; and detailed descriptions of solutions and their associated proofs.

2 BASIC NOTATION

Throughout, we will write T for a finite numerical data type. We think of $T \subseteq \mathbb{R}$, but adding two elements a, b of T can yield a number outside of T, so the *overflow mode* and/or *rounding mode* of T determine the result of the computation $a + b$. For $L, U \in T$ with $L \leq U$, we write $T_{[L,U]} = \{x \in T : L \leq x \leq U\}$. We will consider various implementations of the Bounded Sum function $BS^*_{L,U}$ and $BS^*_{L,U,n}$ on datasets consisting of elements of $T_{[L,U]}$. Except when otherwise stated, BS^* will use the standard Iterated Summation method from Figure 1. In such a case, the functions $BS^*_{L,U}$ and $BS^*_{L,U,n}$ and their sensitivities are completely determined by the choice of overflow and/or rounding modes.

The data types T we will consider in the paper are the following:

- k-bit unsigned integers, whose elements are $\{0, 1, \ldots, 2^k - 1\}$. Standard choices are $k = 32$ and $k = 64$.
- k-bit signed integers, whose elements are $\{-2^{k-1}, -2^{k-1} + 1, \ldots, -1, 0, 1, \ldots, 2^{k-1} - 1\}$. Standard choices are $k = 32$ and $k = 64$.
- (k, ℓ)-bit (normal) floats, which are represented in binary scientific notation as $(-1)^s \cdot (1.M) \cdot 2^E$ with a k-bit mantissa M and an exponent $E \in [-(2^{-\ell-1} - 2), 2^{\ell-1} - 1]$.[6] In 32-bit floats ("singles"), we have $k = 23$ and $\ell = 8$; and in 64-bit floats ("doubles) we have $k = 52$ and $\ell = 11$. In machine learning applications, it is sometimes common to use even lower-precision floating-point numbers for efficiency, such as $k = 7$ and $\ell = 8$ in Google's bfloat16 [45].

We find that for these data types, the implemented sensitivity of Bounded Sum can be much larger than the idealized sensitivity for several reasons, described in the section below.

3 SENSITIVITY LOWER BOUNDS

More rigorous proofs of the lower bounds on sensitivity described in this section can be found in the full version of the paper [10].

Overflow. The default way of dealing with overflow in k-bit integers T (signed or unsigned) is *wraparound*, i.e., the result is computed modulo 2^k. It is immediately apparent how this phenomenon can lead to large sensitivity. If the idealized sum on one dataset u equals the largest element of T, call it $\max(T)$, and on an adjacent dataset u' equals $\max(T) + 1$, then modular arithmetic will yield results that differ by $2^k - 1$. If our parameter settings allow for us to construct two such datasets, then the implemented sensitivity of Bounded Sum will be $2^k - 1$, a completely useless bound because every two numbers of type T differ by at most $2^k - 1$.

In the case of bounded DP on datasets of size n, we can construct two such datasets u and u' if $n \cdot (U - L) \geq 2^k$. As one example of such a pair, consider the following datasets u and u' of unsigned ints T, where we have $L \leq 0$, $U > 0$, and set $n = \left\lceil \frac{\max(T)}{U} \right\rceil + 1$. Let $M = \max(T) - (n-2) \cdot U$. Then, set $u = [U_1, \ldots, U_{n-2}, M, 0]$, and $u' = [U_1, \ldots, U_{n-2}, M, 1]$, where $U_i = U$ for all i.

Then, $BS_{L,U}(u) = \max(T)$, and $BS_{L,U}(u') = \max(T) + 1$, since $M = \max(T) - (n-2) \cdot U$. But because we are using modular arithmetic, we need to evaluate both sums modulo 2^k, in which case $BS^*_{L,U}(u) = \max(T)$, but $BS^*_{L,U}(u') = (\max(T)+1) \mod 2^k = \min(T)$. Hence, $|BS^*_{L,U}(u) - BS^*_{L,U}(u')| = \max(T) - \min(T) = 2^k - 1$. However, because $u \simeq_{bdd} u'$, the idealized sensitivity is $U - L$.

When working with the type T of 64-bit unsigned ints, by setting $L = 0, U = 2^{47}$, and $n = \lceil \max(T)/U \rceil + 1 = 2^{17} + 1$, we get a difference in sums of $2^{64} - 1$, which is more than a factor of 2^{16} larger than these idealized sensitivities. This means, then, that a DP mechanism that claims to offer ε-DP here but calibrates its random distribution to the idealized sensitivity would instead offer no better than $2^{16}\varepsilon$-DP.

In practice, while n may be modest (e.g., $n = 2^{15}$) and much smaller than 2^k, the bounds U and L are typically user-specified parameters. More generally, for example, we can set $L = 0$ and $U = \lceil 2^k/n \rceil$, and we get a sensitivity that is roughly a factor of n larger than the idealized sensitivity $U - L$.

In the case of unbounded DP, n is unconstrained, so we always get an implemented sensitivity of $2^k - 1$, provided that $U > L$. Specifically, there are datasets u and u' of size at most $n = \lceil 2^k/U \rceil$ such that $u \simeq_{unbdd} u'$ and $|BS^*_{L,U}(u) - BS^*_{L,U}(u')| = 2^k - 1$. Again, we get a blow-up in sensitivity of roughly a factor of n. For example, datasets $u = [U_1, \ldots, U_{n-2}, M]$ and $u' = [U_1, \ldots, U_{n-2}, M, 1]$ are adjacent with respect to $\simeq_{unbdd}$. However, we again obtain that $|BS^*_{L,U}(u) - BS^*_{L,U}(u')| = \max(T) - \min(T) = 2^k - 1$, which is much greater than the idealized sensitivity $\max\{|L|, |U|\}$.

Overflow also arises with floating-point arithmetic (leading to results that are $\pm\infty$), but the semantics of arithmetic with $\pm\infty$ are not clearly defined in the IEEE standard, and we consider it better to avoid this possibility entirely (as discussed in our solutions in Section 5).

Rounding. When adding two floating-point numbers, the result may not be exactly representable as a float, but may lie strictly between two adjacent floats. Thus, the actual result is determined by the *rounding mode*. The standard, called *banker's rounding*, rounds the result to the nearest float, breaking ties by rounding to the float whose mantissa has least-significant bit 0. By inspection, when we round a real number z to a (k, ℓ)-bit float $\text{BRound}(z)$, the *relative* effect is minimal. Specifically, $|\text{BRound}(z) - z| \leq |z|/2^{k+1}$.

This may have led to an incorrect impression that floating-point arithmetic is "close enough" to idealized real arithmetic to not cause a significant increase in sensitivity. Unfortunately, we find that this is not the case.

In the case of bounded DP, even a single rounding can cause a substantial blow-up in sensitivity, as we illustrate with the following example. Let Q be a float which is a power of 2, and let $n - 1$ be a power of 2 between 2 and 2^k. Then, we take

$$L = \left(1 + \frac{n-1}{2^{k+1}}\right) \cdot Q,$$
$$U = \left(1 + \frac{n+1}{2^{k+1}}\right) \cdot Q.$$

Thus, L and U are both very close to Q, but their difference is only $U - L = Q/2^k$. Our datasets u and u' will both begin with $n - 1$ copies of L. The iterated sum of these first $n - 1$ elements experiences no rounding, given that all intermediate sums can be represented exactly as (k, ℓ)-bit floats; this is because all these intermediate sums are multiples of 2 raised to the exponent of the floating-point value used to represent these sums, divided by 2^k. The resulting sum of these $(n-1)$ terms is a floating-point number with exponent $\log_2((n-1)Q)$ (since $(n-1)L < 2(n-1)Q$). Thus, the space between adjacent floating-point numbers after this iterated sum is $(n-1)Q/2^k$. Hence, when we add one more copy of L to obtain $BS^*(u)$, the result will lie exactly halfway between two adjacent floats (since L is an odd multiple of $(n-1)Q/2^{k+1}$), and by banker's rounding, will get rounded down. On the other hand, when we add a copy of U to calculate $BS^*(u')$, $BS(u')$ will lie between the same adjacent floats as $BS(u)$, but, by banker's rounding, the result will be rounded up. The effect of these two

[6] Note that there are only $2^\ell - 2$ choices for E. The remaining choices are used to represent *subnormal* floats, as well as $\pm\infty$, and NaN. For simplicity, we only consider normal floats here; the full set of (k, ℓ)-bit floats is considered in the full version of the paper [10].

roundings gives us:

$$BS^*(u') - BS^*(u) = 2\frac{n-1}{2^{k+1}} \cdot Q = (n-1) \cdot (U - L).$$

In contrast, the idealized sensitivity with respect to $\simeq_{bdd}$ is $(U - L)$, so the sensitivity blows up by a factor of $(n-1)$, which is dramatic even for very small datasets. We also show that this construction applies to Kahan summation (contrary to Mironov's hope that it would salvage the sensitivity) and pairwise summation (another common method — the default in `numpy` — where the values are added in a binary tree). Intuitively, this counterexample applies to Kahan summation and pairwise summation because it does not rely on accumulated error (against which Kahan and pairwise summation could protect), but on the error inherent to rounding the sum to a k-bit float.

Repeated Rounding. The above example does not give anything interesting for unbounded DP, since the idealized sensitivity is $\max\{|L|, |U|\}$, and $BS^*(u') - BS^*(u) \leq Q < U$. However, we can obtain a sensitivity blow-up by exploiting the accumulated effect of *repeated rounding*. Consider a dataset whose first $(n/2)$ elements are U. After that, each rounding can have the effect of increasing the sum by $\Theta(nU/2^k)$, for a total rounding error of $\Theta(n^2U)/2^k$. We show how to exploit this phenomenon to construct two datasets $u \simeq_{unbdd} u'$ which differ on their middle element such that

$$\left|BS^*_{L,U}(u) - BS^*_{L,U}(u')\right| \geq \left(1 + \Omega\left(\frac{n^2}{2^k}\right)\right) \cdot U.$$

Thus, the sensitivity blows up by a factor of $\Theta(n^2/2^k)$. This allows us to exhibit a sensitivity blow-up with datasets of size $n = \Theta(2^{k/2})$, which are plausible even for 64-bit floats (where $k = 52$) and quite easy to obtain for 32-bit and lower-precision floats.

This implemented sensitivity can be obtained with the following two datasets. Let U be a power of 2, m an integer power of 2, $n = 2m$, and

$$L = -\left(\frac{U \cdot m}{2^k}\right) \cdot \left(\frac{1}{2} - \frac{1}{2^k}\right).$$

Additionally, let

$$u = [U_1, \ldots, U_m, x_1, L_2, x_3, L_4, \ldots, x_{m-1}, L_m],$$

$$u' = [U_1, \ldots, U_{m-1}, x_1, L_2, x_3, L_4, \ldots, x_{m-1}, L_m],$$

where $U_i = U$, $L_i = L$, and $x_i = x$ for all i, and

$$x = \left(\frac{U \cdot m}{2^k}\right) \cdot \left(\frac{1}{2} + \frac{1}{2^k}\right).$$

Note that u and u' are equivalent with the exception that u' contains $m - 1$ copies of U rather than m copies of U, so $u \simeq_{unbdd} u'$. All intermediate sums computed in the calculation of $BS^*_{L,U}[U_1, \ldots, U_m]$ can be represented exactly as (k, ℓ)-bit floats; this is because all these intermediate sums are multiples of 2 raised to the exponent of the floating-point value used to represent these sums, divided by 2^k. The ability to represent these intermediate sums exactly implies that

$$BS^*_{L,U}[U_1, \ldots, U_{m-1}] = BS_{L,U}[U_1, \ldots, U_{m-1}] = (m-1) \cdot U$$

and

$$BS^*_{L,U}[U_1, \ldots, U_m] = BS_{L,U}[U_1, \ldots, U_m] = m \cdot U.$$

However, $BS_{L,U}[U_1, \ldots, U_m, x] = m \cdot U + x$ is not exactly representable as a (k, ℓ)-bit float and will be rounded up to $m \cdot U + mU/2^k$. Continuing with the computation of $BS^*_{L,U}(u)$, when we add L, the resulting sum is not large enough to escape rounding down by banker's rounding. The same reasoning applies to all subsequent additions of x and L, where adding x has the effect of adding $mU/2^k$ and adding L has the effect of adding 0. This yields a total sum of

$$BS^*_{L,U}(u) = m \cdot U + \frac{m^2 \cdot U}{2^{k+1}}.$$

In the case of u', we have one less U term, so we consider the sum $BS_{L,U}[U_1, \ldots, U_{m-1}, x] = (m-1) \cdot U + x$. This is again not a representable (k, ℓ)-bit float, and so $BS^*_{L,U}[U_1, \ldots, U_{m-1}, x] = (m-1) \cdot U + (m-1)U/2^k$. When we add L, the closest representable float to the sum is $(m-1) \cdot U$, and so by banker's rounding the sum gets rounded down to this value. Then, for u', adding x and L in succession has the effect of adding 0. In total, this yields

$$BS^*_{L,U}(u') = (m-1) \cdot U.$$

Therefore,

$$|BS^*_{L,U}(u) - BS^*_{L,U}(u')| = \frac{m^2 \cdot U}{2^{k+1}} + U = \frac{n^2 \cdot U}{2^{k+3}} + U,$$

since $n = 2m$. This is a factor $n^2/2^{k+3} + 1$ larger than the idealized sensitivity $\max\{|L|, |U|\} = U$.

By setting $k = 52$ (as is the case for 64-bit floats), $L = -2^{-23} \cdot (\frac{1}{2} + 2^{-52})$, $U = 1$, and $n = 2^{30}$, we get a difference in sums of $2^5 + U = 33$, which is a factor 33 larger than the idealized sensitivity $\max\{|L|, |U|\} = 1$.

Reordering. Finally, we exhibit potential vulnerabilities based on ambiguity about whether datasets are ordered or unordered. In the DP theory literature, datasets are typically considered unordered (i.e., multisets) when using unbounded DP. (Indeed, adjacency is often described by requiring that that the *symmetric difference* between the multisets u and u' has size at most 1, or by requiring that the histograms of u and u' have ℓ_1 distance at most 1.) On the other hand, when using bounded DP, it is common to denote datasets as ordered n-tuples. (Indeed, adjacency is often described by requiring that the *Hamming distance* between the n-tuples is at most 1.) Most implementations of DP are not explicit about these choices, but the wording in the documentation suggests the same conventions (e.g., see Section 1.3 in [47]).

In theory, the distinction does not matter much, because most of the functions we compute (such as Bounded Sum) are symmetric functions and do not depend on the ordering.

However, when implementing these functions using finite data types, the ordering can matter a great deal. Indeed, we show that there are datasets u and u' where u' is a permutation of u (so define exactly the same multisets) but

$$\left|BS^*_{L,U}(u) - BS^*_{L,U}(u')\right| \geq \left(1 + \Omega\left(\frac{n^2}{2^k}\right)\right) \cdot U.$$

That is, we can obtain the same kind of blow-ups in sensitivity that we obtained due to rounding by instead just reordering the dataset. (Our rounding attacks preserved order, i.e., we obtained u' by either changing one element of the ordered tuple u, or by

inserting/deleting one element of u without changing the other elements.)

This implemented sensitivity can be obtained with the following two datasets. Let $L = 2^j$ for some integer j, let $U = 2^{j+1}$, and let

$$u = [L_1, \ldots, L_{2^{(k+1)}}, U_1, \ldots, U_{2^k}],$$
$$u' = [U_1, \ldots, U_{2^k}, L_1, \ldots, L_{2^{(k+1)}}].$$

Note that u' is a permutation of u. The first 2^{k+1} terms of u can be added exactly; i.e.,

$$BS^*_{L,U}[L_1, \ldots, L_{2^{(k+1)}}] = 2^{k+1} \cdot L.$$

The next 2^k terms of u can also be added exactly, and so

$$BS^*_{L,U}(u) = 2^{k+1} \cdot L + 2^k \cdot U.$$

Likewise, the first 2^k terms of u' can be added exactly, and hence $BS^*_{L,U}(u') = 2^k \cdot U$. However, the addition of the next 2^{k+1} L terms to u' yield intermediate sums which are *not* representable exactly as (k, ℓ)-floats. For every L term that we add, the sum falls exactly in the middle of two adjacent floats, and since the corresponding mantissa ends with an even bit, by the definition of banker's rounding the sum will round down to $2^k \cdot U$ at each step. Therefore, we get a final sum of

$$BS^*_{L,U}(u') = 2^k \cdot U.$$

Therefore,

$$|BS^*_{L,U}(u) - BS^*_{L,U}(u')| = 2^{k+1} \cdot L = 2^k \cdot U.$$

Under $\simeq_{unbdd}$, the idealized sensitivity is $\max\{|L|, |U|\} = U$, and so the implemented sensitivity is a factor 2^k larger than the idealized sensitivity. Under $\simeq_{bdd}$, the idealized sensitivity is $U - L$, and so the implemented sensitivity is a factor $2^k \cdot U/(U - L)$ larger than the idealized sensitivity.

As a concrete example, if we set $j = 0$ (so $L = 1$ and $U = 2$), then for 32-bit floats (i.e., $k = 23$) we get datasets of length $2^{24} + 2^{23}$ and a difference in sums of $2^{23} \cdot U = 2^{24}$, which is a factor 2^{23} larger than the idealized sensitivity $\max\{|L|, |U|\} = 2$ under $\simeq_{unbdd}$ and a factor 2^{24} larger than the idealized sensitivity $U - L = 1$ under $\simeq_{bdd}$.

One reason this issue may have been missed before is that floating-point arithmetic is *commutative*, e.g., $\text{BRound}(a + b) = \text{BRound}(b + a)$, so it may seem like order does not matter. However, *associativity* is required for the sum to be invariant under arbitrary permutations, which fails for floating-point arithmetic due to rounding.

Modular integer arithmetic is associative, so this issue does not arise for k-bit (unsigned or signed) integers with wraparound. However, if instead we use *saturation* arithmetic, where values get clamped to the range $[\min(T), \max(T)]$ (this addresses the aforementioned sensitivity problems with wraparound), then addition of *signed* integers is no longer associative. Indeed, if $n \cdot \min\{U, -L\} \geq 2^{k+1}$, we exhibit two datasets u, u' of length n such that u' is a permutation of u and

$$\left|BS^*_{L,U,n}(u) - BS^*_{L,U,n}(u')\right| \geq 2^k - 1.$$

That is, we get no improvement over the (trivial) sensitivity we had with modular arithmetic.

This implemented sensitivity can be achieved with the following two datasets. Set $\alpha = \lceil \frac{\max(T) - \min(T)}{|L|} \rceil$, $\beta = \lceil \frac{\max(T) - \min(T)}{U} \rceil$, and let

$$u = [L_1, \ldots, L_\alpha, U_1, \ldots, U_\beta],$$
$$u' = [U_1, \ldots, U_\beta, L_1, \ldots, L_\alpha],$$

where $U > 0$ and $L < 0$. This last condition is crucial, given that the idea behind the attack is to play with the positive and negative signs to exploit the fact that non-associativity of saturation arithmetic only applies when the result of an intermediate sum falls outside the saturation bounds (e.g., outside of $\min(T)$ and $\max(T)$). First, by definition of α, we see that $BS^*_{L,U}[L_1, \ldots, L_\alpha] = \min(T)$, given that saturation arithmetic clamps the negative values at $\min(T)$. When we add the next β "U" terms, by definition of β, the intermediate sums go all the way up to $\max(T)$, and then saturation arithmetic clamps the sum at $\max(T)$, since all of the U terms are positive. Therefore,

$$BS^*_{L,U}(u) = \max(T).$$

Likewise, for u', the first β "U" terms result in an intermediate sum of $\max(T)$, and then the sum remains clamped there. The next α "L" terms then result in an intermediate sum of $\min(T)$, where it remains clamped. Therefore,

$$BS^*_{L,U}(u') = \min(T).$$

The result is a difference in sums of

$$\left|BS^*_{L,U,n}(u) - BS^*_{L,U,n}(u')\right| = \max(T) - \min(T) = 2^k - 1,$$

and since u' is a permutation of u, we have $u \simeq u'$. Under $\simeq_{unbdd}$ the idealized sensitivity is $\max\{|L|, |U|\}$, and under $\simeq_{bdd}$, the idealized sensitivity is $U - L$, so this implemented sensitivity can be much greater than the idealized sensitivities.

As a concrete example, if we set $L = -2^{14}$, $U = 2^{15}$, and $k = 32$, then the above construction gives us two datasets of size $n = \lceil \frac{\max(T) - \min(T)}{|L|} \rceil + \lceil \frac{\max(T) - \min(T)}{U} \rceil$ where the difference in sums is $2^k - 1 = 2^{32} - 1$. This is more than a factor 2^{16} larger than the idealized sensitivity $\max\{|L|, |U|\} = 2^{15}$ under $\simeq_{unbdd}$ and the idealized sensitivity $U - L = 2^{15} + 2^{14}$ under $\simeq_{bdd}$.

4 ATTACKS

We show that our attacks can be carried out on many existing implementations of differential privacy. In each attack, we set the privacy-loss parameter to $\varepsilon \in \{0.5, 1\}$, set the parameters L, U, and possibly n as required by our sensitivity lower bounds, construct two appropriate adjacent datasets $u \simeq_{bdd} u'$ or $u \simeq_{unbdd} u'$, and run the supposedly ε-DP Bounded Sum mechanism $\mathcal{M}$ on u and u'. We show that by applying a threshold test to the outputs, we can almost perfectly distinguish $\mathcal{M}(u)$ and $\mathcal{M}(u')$. For example, in one experiment, we succeed in correctly identifying whether the dataset is u or u' in all but 3 out of 20,000 runs of the mechanism, which we show would be astronomically unlikely for a mechanism that is truly 0.5-DP. These attacks are described in more detail in the full version of the paper [10].

Our overflow attack on integers works on IBM's diffprivlib.

	Adjacency relation	Implemented sensitivity	Conditions
Idealized	Bounded DP	$U - L$	
	Unbounded DP	U	
Modular addition (signed or unsigned `ints`)	Bounded DP	$\max(T) - \min(T)$	$nU > \max(T)$
	Unbounded DP	$\max(T) - \min(T)$	
Saturation (signed `ints`)	Bounded DP	$\max(T) - \min(T)$	$nU > \max(T)$ $U > 0$ and $L < 0$
	Unbounded DP	$\max(T) - \min(T)$	$U > 0$ and $L < 0$
Floats with k-bit mantissa	Bounded DP	$\geq (n-1) \cdot (U - L)$	$U \geq 2^k \cdot (U - L)$
	Unbounded DP	$(1 + \Theta(n^2/2^k)) \cdot U$	$L \leq -U \cdot n/2^{k+3}$

Table 1: Lower bounds obtained in our attacks for numerical type T and $-U \leq L \leq U$.

Our rounding attack on floats works on Google's DP library, IBM's diffprivlib, OpenDP / SmartNoise,[7] and Chorus. Our repeated rounding attacks on floats work on Google's DP library, IBM's diffprivlib, OpenDP / SmartNoise,[8] Chorus, and PINQ.

Lastly, our re-ordering attack works on Google's DP library, IBM's diffprivlib, OpenDP / SmartNoise, Chorus, and PINQ. Chorus only offers sensitivities for the bounded DP setting, so slight adjustments were made for the repeated rounding attacks — specifically, a value of 0 was appended to dataset v. With the exception of OpenDP, none of the libraries include a disclaimer indicating that the integer overflow or floating-point rounding behaviors we exploit could yield vulnerabilities.

Our attacks utilize contrived datasets u and u' that are unlikely occur "in the wild." However, our overflow and rounding attacks can easily be applied to attack realistic datasets given access to a DP system that fields external queries with user-defined microqueries (a.k.a., mappers or row transforms). Specifically, consider a dataset $x = [x_1, \ldots, x_n]$ where each record x_i contains a known/public user identifier uid_i for the i'th individual, and a sensitive bit val_i. Suppose further that the dataset is sorted by the identifiers, i.e., $\text{uid}_1 \leq \text{uid}_2 \leq \cdots \leq \text{uid}_n$. Then it is straightforward for an adversarial analyst to define a simple micro-query q such that

$$q(x) = [q(x_1), \ldots, q(x_n)] = \begin{cases} u & \text{if } \text{val}_i = 1 \\ u' & \text{if } \text{val}_i = 0 \end{cases}$$

where u, u' are the datasets in our attacks, and they differ on the i'th record. Thus, by seeing the result of the bounded-sum mechanism on $q(x)$, the adversary can extract the i'th individual's bit val_i with very high probability. This can be generalized to not just attack a particular individual, but a constant fraction of the individuals in the dataset (e.g., any individual in the middle half of the dataset). Note that our use of microqueries here is different than in Haeberlen et al. [22]. Our microqueries are pure functions, not making use of any side channels or global state, and are simple enough to be implemented in virtually any domain-specific language (DSL) for microqueries. Indeed, see Figure 3 for a code snippet showing how our attack would look in SQL and Figure 4 for PINQ.

[7] Our paper attacks the code at https://github.com/opendp/opendp, and the attacks also work on OpenDP/SmartNoise-core at https://github.com/opendp/smartnoise-core.
[8] In particular, we are able to attack `opendp.trans.make_sized_bounded_sum` and `opendp.trans.make_bounded_sum`.

```
SELECT
    SUM(CASE
        WHEN uid < {m} THEN {U}
        WHEN uid = {m} THEN val * {U}
        WHEN uid % 2 = 1 THEN {pos_val}
        ELSE {neg_val}
    END)
FROM unsized_64_u
```

Figure 3: Example SQL code for our repeated rounding attack. Here the upper clamping bound is `U`, the lower clamping bound is `L=-U`, `uid` denotes the user id attribute of an individual record, `m` is the middle user id (the one we wish to attack), `val` is the individual's sensitive bit, and `pos_val` and `neg_val` are particular floats in the interval `[L, U]` from our attack.

```
var dp_sum = new PINQueryable<string>(arr_v_queryable,
     new PINQAgentLogger(filepath))
    .Select(l => l.Split(','))
    .Select(terms => new Tuple<long, double>(Convert.
     ToInt64(terms[0]), Convert.ToDouble(terms[1])))
    .Select(tup => (tup.Item1 < attackUID) ? U :
        (tup.Item1 == attackUID) ? U * tup.Item2 :
        (tup.Item2 % 2 == 1) ? pos_val : neg_val)
    .NoisySum(100.0, v => v);
```

Figure 4: Example PINQ code for our repeated rounding attack.

Since PINQ uses 64-bit floats, our repeated rounding attack requires quite a large dataset (e.g., $n = 2^{28}$), and we were not able to complete execution due to memory timeouts, but the attack is possible in principle. Perhaps due to concerns about timing and other side-channel attacks (see Section 1.2), recent DP SQL systems such as Chorus do not support user-defined microqueries. However, it may be still be possible to carry out our basic rounding attack (not the repeated rounding one), since it can be implemented in such a way that the microquery is simply the clamp function (applied automatically in most implementations of Noisy Bounded Sum), if we know that the dataset is sorted by the value we wish to attack (e.g., consider a dataset of salaries, and where the employee with second-highest salary wishes to find out the CEO's salary). Finally, we also hypothesize that the attack can be carried out on DP Machine Learning systems that use DP-SGD, using a user-defined loss

function $\ell(\theta, x_i)$, designed so that $\nabla_\theta \ell(\theta, x_i)$ equals our desired microquery $q(x_i)$. If we set parameters so that there is only one iteration, using the entire dataset as a batch, then the output parameter vector θ will exactly give us a Noisy Bounded Sum. Since these ML systems allow using low-precision floats, these attacks should be feasible even on very small datasets.

To carry out our reordering attacks on a DP query system, we would need a system that can be made to perform a data-dependent reordering. These attacks indicate that we need to be explicit and careful about how we treat ordering when we implement DP.

In any case, our attacks and experimental results reported in the appendix already demonstrate that essentially all of the implementations of DP fail to meet their promised DP guarantees.

Responsible Disclosure. Immediately after the submission of this paper, we shared the paper with the maintainers of all of the DP libraries that exhibit the vulnerabilities we described and informed them that we would wait 30 days to post the paper publicly, to give them time to implement any needed patches (like our solutions below). In response, all offered some indication that they are working to resolve these issues, and Google's Bug Hunter program acknowledged our contribution to Google's security with an Honorable Mention.[9]

All of the authors of this paper are members of the OpenDP team and are involved with the development of the library. The findings of this paper occurred while the authors were writing mathematical proofs to accompany the algorithms that are part of the OpenDP library as part of our vetting process, upon which we realized the vulnerabilities described in this paper. We are updating the OpenDP library following the roadmap described in Section 6.[10] We believe that our findings also illustrate the importance of vetting processes such as the one put in place for OpenDP.

Code. Code for our attacks and experiments is available in the Github repository at https://github.com/cwagaman/underestimate-sensitivity.

5 SOLUTIONS

Dataset Adjacency Relations. Toward addressing the potential reordering vulnerabilities, we propose that when datasets $u = [u_1, \ldots, u_n]$ are stored as ordered tables, we should define and distinguish four adjacency relations $\simeq$. Specifically, the bounded-DP relation $\simeq_{bdd}$ should separate into the usual $\simeq_{Ham}$ ("Hamming"), where $u \simeq_{Ham} u'$ means that there is at most one coordinate i such that $u_i \neq u'_i$, and $\simeq_{CO}$ ("change-one"), where $u \simeq_{CO} u'$ means that we can convert the multiset of elements in u into the multiset of elements in u' by changing one element. Equivalently, $u \simeq_{CO} u'$ iff there is a permutation π such that $\pi(u) \simeq_{Ham} u'$. Similarly, the unbounded-DP relation $\simeq_{unbdd}$ should split into an ordered version $\simeq_{ID}$ ("insert-delete") and an unordered version $\simeq_{Sym}$ ("symmetric distance"). Throughout the full version of the paper [10], all of our theorems clearly state the adjacency relation that is being used. By being explicit about ordering in our adjacency relation, and in particular analyzing DP and sensitivity with respect to a specific relation, we can avoid the reordering vulnerabilities. In particular, a DP system using ordered adjacency relations should take care to disallow data-dependent reorderings, unless they can be shown to preserve ordered adjacency (or, more generally are stable with respect to Hamming distance or insert-delete distance).

Random Permutations (RP). Given the above adjacency relations, it still remains to find implementations of Bounded Sum that achieve a desired sensitivity with respect to them. In general, it is easier to bound sensitivity with respect to the ordered relations $\simeq_{Ham}$ and $\simeq_{ID}$. For example, we can show that Iterated Sum of signed integers with saturation arithmetic has the idealized sensitivities of $U - L$ and $\max\{U, |L|\}$ with respect to $\simeq_{Ham}$ and $\simeq_{ID}$, respectively, whereas our reordering attacks show that the sensitivities can be as large as $2^k - 1$ with respect to $\simeq_{CO}$ and $\simeq_{Sym}$.

Motivated by this observation, we give a general method for converting sensitivity bounds with respect to the ordered relations into the same bounds with respect to the unordered relations: randomly permute the dataset before applying the function. To formalize the effect of this transformation, we need to extend the definition of sensitivity to randomized functions. Following [43], we say that a randomized function f has *sensitivity at most* Δ *with respect to* $\simeq$ if for all pairs of datasets u, u' such that $u \simeq u'$, there is a *coupling* of the random variables $f(u)$ and $f(u')$ such that $\Pr[|f(u) - f(u')| \leq \Delta] = 1$. We prove that if RP is the random permutation transformation on datasets and f is any function on datasets, we have:

$$\begin{aligned} \Delta_{Sym}(f \circ RP) &\leq \Delta_{ID} f, \text{ and} \\ \Delta_{CO}(f \circ RP) &\leq \Delta_{Ham} f. \end{aligned}$$

Given this result, it suffices for us to obtain sensitivity bounds with respect to ordered adjacency relations.

Checking or Bounding Parameters. Many of our attacks only lead to a large increase in sensitivity under certain parameter regimes, e.g., $n \cdot U \geq 2^k$. In the case of bounded DP, all of these parameters (n, U, L, k) are known in advance (before we touch the sensitive dataset), and we can prevent the problematic scenarios by either constraining the parameters or incorporating the dependence on those parameters into our sensitivity bounds. Indeed, we show that many of the sensitivity lower bounds discussed in Section 3 are tight by giving nearly matching upper bounds.

For example, in the case of integer data types with bounded DP, we prove that the implemented sensitivity equals the idealized sensitivity provided that $U \cdot n \leq \max(T)$ and $L \cdot n \geq \min(T)$. These conditions ensure that overflow cannot occur. Since for bounded DP, U, L, and n are public parameters that do not depend on the sensitive dataset, we can simply check that these conditions hold before performing Bounded Sum.

For floats, we provide a similar-style parameter check to ensure that a summation does not hit `±inf`.

Truncated Summation. For the case of unbounded DP, we cannot perform a parameter check involving n as above, since n is not publicly known and may be sensitive information. We can, however, achieve a similar effect by composing a solution for bounded DP with a *truncation* operation on datasets, namely

$$Trunc_n(u) = [u_1, u_2, \ldots, u_{\min\{n, \text{len}(u)\}}].$$

[9] https://bughunters.google.com/profile/d946f172-9bd8-4b84-9f17-d86046f5af11.
[10] For example, see https://github.com/opendp/opendp/pull/465 and https://github.com/opendp/opendp/pull/467.

This truncation operation behaves nicely with respect to the *ordered* adjacency relation $\simeq_{ID}$. Specifically, we prove that for every function f on datasets,

$$\Delta_{\simeq_{ID}}(f \circ Trunc_n) \leq \max\left\{\Delta_{\simeq_{ID},\leq n} f, \Delta_{\simeq_{CO},\leq n} f\right\}, \tag{2}$$

where $\Delta_{\simeq,\leq n}$ denotes sensitivity restricted to datasets $u \simeq u'$ of length at most n. Intuitively, inserting or deleting an element from a dataset u either results in inserting or deleting an element from $Trunc_n(u)$ (if no truncation occurs) or changing one element of u (if truncation occurs).

Applying (2) to the case of truncated integer summation, where f is Iterated Summation, we recover the idealized sensitivity with respect to $\simeq_{ID}$ provided that $n \cdot U \leq \max(T)$ and $n \cdot L \geq \min(T)$ (to prevent overflow) and both U and L are of the same sign (so that $U - L \leq \max\{U, |L|\}$ and the idealized sensitivity with respect to $\simeq_{CO}$ is no larger than the idealized sensitivity with respect to $\simeq_{ID}$).

Split Summation. The above example of truncated integer summation with respect to unbounded DP is one of several cases where we obtain better sensitivity bounds when U and L are of the same sign. Another is integer summation with saturation arithmetic with respect to unordered adjacency relations: when U and L are of different signs, this is vulnerable to our reordering attack, but when they are of the same sign, we show that it has implemented sensitivity equal to the idealized sensitivity. We also give an example below with floating-point numbers where computing sums on terms with matching signs helps achieve an implemented sensitivity that is closer to the idealized sensitivity.

To take advantage of the benefits that can come from summing terms with the same sign, we introduce the *split summation* technique, where we separately sum the positive numbers and the negative numbers in the dataset. That is, given a dataset u and a base summation method BS^*, we let $pos(u)$ be the dataset consisting of the positive elements of u, $neg(u)$ be the dataset consisting of the negative elements of u, and define

$$BS^{**}_{L,U}(u) = BS^*_{0,U}(pos(u)) + BS^*_{L,0}(neg(u)).$$

Importantly, adding or removing an element from a dataset u corresponds to adding or removing an element from only one of $pos(u)$ and $neg(u)$, so we do not incur a factor of 2 blow-up in sensitivity. Using this split summation technique in combination with either truncation or saturation arithmetic allows us to recover the idealized sensitivity for summation of signed integers with unbounded DP.

Sensitivity from Accuracy. Our matching upper bound on the sensitivity of Iterated Summation of floats with banker's rounding is obtained via reduction to accuracy. Specifically, we can use the triangle inequality to show that we have

$$|BS^*(u) - BS^*(u')| \leq \\ |BS(u) - BS(u')| + |BS^*(u) - BS(u)| + |BS^*(u') - BS(u')|.$$

The first term is bounded by the idealized sensitivity, and the latter two terms can be bounded by using known numerical analysis results about the accuracy of iterated summation. Specifically, a bound from Wilkinson [46] shows that

$$|BS^*(u) - BS(u)| = O\left(\frac{n}{2^k}\sum_{i=1}^{n}|u_i|\right) = O\left(\frac{n^2 \cdot \max\{|L|,|U|\}}{2^k}\right).$$

Note that the resulting upper bound on sensitivity described above has an "n" term, so it can only be applied directly in the case of bounded DP. To handle unbounded DP, we can combine it with the truncation technique described above.

Specifically, combining Iterated Summation BS^* with the Truncation transformation $Trunc_n$, we get an unbounded-DP sensitivity bound of:

$$\Delta_{\simeq_{ID}}\left(BS^*_{L,U} \circ Trunc_n\right) \leq \left(1 + O\left(\frac{n^2}{2^k}\right)\right) \cdot \max\{|L|,|U|\}, \tag{3}$$

provided that L and U have the same sign, and similarly for the unordered sensitivity $\Delta_{\simeq_{Sym}}$ if we also combine with a random permutation. Thus, when $n \ll 2^{k/2}$, we recover almost the idealized sensitivity bound of $\max\{|L|,|U|\}$. Higham [24] has proven that other summation methods for floats, such as Pairwise Summation (which is the default in `numpy`) and Kahan Summation have better bounds on accuracy, allowing us to replace the $O(n^2)$ above with $O(n \log n)$ or $O(n)$, respectively. These methods closely approximate the idealized sensitivity whenever $n \ll 2^k$, which covers many practical scenarios (see discussion in Section 6).

Shifting Bounds. For the case of bounded DP, some of the sensitivity blow-ups (such as the one exhibited in the basic rounding attack) come from having U (or $\max\{|L|,|U|\}$) rather than $U-L$ in the implemented sensitivity bound, as U can be much larger than $U-L$. One way to address this is to subtract L from every element of the dataset, so that all elements lie in the interval $[L' = 0, U' = U - L]$, apply our solutions that have U' in the sensitivity bound, and then add $L \cdot n$ at the end as post-processing. That is, we convert a method BS^* with sensitivity depending on $\max\{|L|,|U|\}$ (such as Expression (3)) into a method BS^{**} with sensitivity depending on $U - L$ as follows:

$$BS^{**}_{L,U,n}(u) = BS^*_{0,U-L,n}(u_1 - L, u_2 - L, \ldots, u_n - L) + L \cdot n,$$

where a noisy version of the $BS^*_{L,U,n}$ term is computed first and $L \cdot n$ is added as a post-processing step, after noise addition. In particular, combining this technique with the Sensitivity-via-Accuracy analysis of Iterated Summation, we obtain a bounded-DP sensitivity bound of

$$\Delta_{\simeq_{Ham}} BS^{**}_{L,U,n}(u) \leq \left(1 + O\left(\frac{n^2}{2^k}\right)\right) \cdot (U - L).$$

Reducing Floats to Ints. Another attractive solution for floating-point summation is to reduce it to integer summation, since the latter achieves the idealized sensitivity with simple solutions. The idea behind this method is to cast floats to fixed-point numbers, which we can think of as k-bit integers. To achieve this casting, we introduce a discretization parameter D (which is chosen by the data analyst and corresponds to the precision with which numbers are represented – e.g., setting $D = 0.01$ means that values are represented to the hundredths place), round each of the dataset elements according to the discretized interval, and apply one of our integer solutions. We can think of this process of "integerizing"

as a dataset transform. This idea of mapping floating-point values to integers is similar to the technique of performing *quantization* to work with low-precision number formats in machine learning applications [33].

To make use of the full range of integers available to us, we first center each term; that is, we subtract the midpoint of U and L, namely $(U+L)/2$, from every term. Then, we divide each resulting floating-point value by some (small) analyst-specified discretization parameter D; and we then round the result to the nearest integer. For some intuition, a smaller value of D means that the sum is being computed with higher precision since fewer distinct floating-point values will map to the same integer.

We provide a complete description and analysis of this solution in the full version of the paper. To simplify the description here, we present the solution in the specific case where $L = -U$. In the general case, it will be necessary to shift the discretization range so that the mid-point is $M = (U+L)/2$, in order to take full advantage of the range of signed integers and obtain improved accuracy as compared to the no-shift strategy.

Before describing this method formally, we provide some intuition. The idea behind this strategy is to group floating-point values into buckets (where close values are put into the same bucket or close buckets) and enumerate the buckets. More specifically, given the bounds L and U for our floats and the discretization parameter D, where $L = -U$ and $K = \lfloor U/D \rfloor$, we round all elements of the interval $[L, U]$ to the nearest element of the sequence

$$-KD, -(K-1)D, \ldots, -D, 0, D \ldots, KD,$$

so $|L| \approx U \approx KD$. We then use the signed integers to enumerate this sequence. If $K \leq (2^{k-1} - 1)$, we can think of the rounded elements as corresponding to k-bit integers, which we can then sum using a summation method for integers. We can then post-process the resulting (noisy) sum into a (noisy) floating-point sum. We discuss the optimal choice of the discretization parameter D after the description of the method.

Formally, we can consider the row-by-row transformation mapping a dataset $u = [u_1, \ldots, u_n]$ of floats to the signed integer dataset

$$\mathit{Float2Int}_{L,U,D}(u) = [\mathit{round}(u_1/D), \ldots, \mathit{round}(u_n/D)],$$

where $\mathit{round}(\cdot)$ denotes rounding to the nearest integer (we further explore the role of the $\mathit{round}(\cdot)$ function below). We can then apply a summation method $BS^*_{-K,K}$ for k-bit integers, and then rescale and shift to obtain our floating-point result:

$$BS^{**}_{L,U,D}(u) = (BS^*_{-K,K} \circ \mathit{Float2Int}_{L,U,D})(u) \cdot D. \tag{4}$$

The sensitivity of (4) can be bounded: as long as we apply one of our solutions for integers, the Bounded Sum of the integers will have the idealized sensitivity. However, to bound the sensitivity of the overall function, we would still need to account for the potential rounding that can occur when multiplying the integer sum by D.

An even better approach for the summation than (4) is to perform noise addition and obtain DP *before* scaling back to floats. That is, we consider the DP mechanism

$$\mathcal{M}_{L,U,D}(u) = \left((BS^*_{-K,K} \circ \mathit{Float2Int}_{L,U,D})(u) + \mathit{Noise}(K/\varepsilon)\right) \cdot D,$$

where $\mathit{Noise}(s)$ denotes a noise distribution with scale s suitable for k-bit integers (e.g., the discrete Laplace Mechanism [20]). Then the privacy of $\mathcal{M}$ follows from the privacy of the noisy integer summation together with the post-processing property of differential privacy, and there is no need to analyze any floating-point rounding effects in either the sensitivity analysis or the noise addition step. Indeed, implementing discrete noise-addition mechanisms is much simpler than implementing floating-point noise-addition mechanisms (such as Mironov's Snapping Mechanism [37]).

One remaining question is how to pick the discretization parameter D to maximize accuracy. A choice that maintains high precision and reduces the risk of (still-private) answers that overflow is

$$D = (U-L) \cdot n_{\max}/(2^{m-2} - n_{\max}),$$

where $n_{\max}$ is the maximum expected dataset size and m is the bitlength of the integers we are using (e.g., $m = 64$).

This choice of D ensures that $K \cdot n_{\max} < 2^m/4$; note that $K \cdot n_{\max}$ is the largest attainable value by the sum. Therefore, all possible integer sums (before noise addition) are contained in the middle 50% of the set of signed integers, which means that the noisy integer summation is very unlikely to experience overflow/saturation. (Note that, although we are guaranteed to avoid overflow and saturation in the deterministic summation as long as we perform appropriate parameter checks, we also want to avoid overflow and saturation in the noisy summation to offer statistically useful outputs from the noisy summation.) For example, when working with the discrete Laplace Mechanism, by analyzing the tail bounds of the distribution, the probability of overflow for the Noisy Bounded Sum can be shown to be $\exp(-\Omega(K \cdot n_{\max}/(K/\varepsilon))) = \exp(-\Omega(\varepsilon \cdot n_{\max}))$, which will be astronomically small for typical settings of parameters.

We should also analyze the impact of the rounding in *Float2Int* on accuracy. The rounding affects the sum of the elements in $\mathit{Float2Int}_{L,U,D}(u)$ by at most $\pm n/2$ (since, when rounding to the nearest integer, u_i/D can get rounded by at most $\pm 1/2$, and we add up n elements) on a worst-case dataset of size n, and by $\pm O(\sqrt{n})$ on typical datasets (since we expect elements to be equally likely to be rounded up as rounded down). If we use randomized rounding for the $\mathit{round}(\cdot)$ function, then we can obtain an accuracy of $\pm O(\sqrt{n})$ even for worst-case datasets. Scaled back to floating point numbers, this yields a final accuracy impact of at most $D \cdot n/2 = O((U-L) \cdot n \cdot n_{\max}/2^m)$ on worst-case datasets and $O((U-L) \cdot \sqrt{n} \cdot n_{\max}/2^m)$ on typical datasets. These errors are comparable to the accuracy bounds for non-private iterated summation of (k, ℓ)-bit floats [24], with the advantage of not affecting the sensitivity or privacy analysis.

This solution is a variant of the common suggestion to replace floating-point arithmetic in DP libraries with integer or fixed-point arithmetic [3, 18]. However, for a data analyst, our solution retains the usability benefits of floating-point numbers, such as the large dynamic range afforded by the varying exponents. Indeed, the input dataset and output result remain as floating-point numbers, and the conversion to and from integer/fixed-point representation only happens internal to the (noisy) sum. In particular, the mapping between floating-point numbers and fixed-point numbers is determined dynamically based on the parameters L and U, in contrast to adopting a single fixed-point representation throughout a DP library.

Modular Sensitivity. One way to address the large sensitivity of Bounded Sum over k-bit integers with wraparound is to change our definition of sensitivity, measuring the distance between k-bit integers as if they are equally spaced points on a circle. That is, we replace $|f(u) - f(u')|$ in the definition of sensitivity with $\min\{|f(u) \ominus f(u')|, |f(u') \ominus f(u)|\}$, where $\ominus$ is subtraction with wraparound over the integer type T on which Bounded Sum is being computed; we call this the *modular sensitivity* of f. With this change, Bounded Sum recovers its idealized sensitivity. But this begs the question of whether functions with bounded modular sensitivity can still be estimated in a DP manner. Fortunately, we show that the answer is *yes*: if we add *integer-valued* noise (such as in the Discrete Laplace [20] or Discrete Gaussian [8] Mechanisms) and also do the noise addition with wraparound, then the result achieves the same privacy parameters as if we had done everything exactly over the integers, with no wraparound. Indeed, by the fact that modular reduction is a ring homomorphism, we can analyze the output distribution as if we had done modular reduction only at the end, which amounts to post-processing.

Several DP libraries already implement this solution, because it happens by default when everything is a k-bit integer. The reason we were able to attack IBM's diffprivlib with the overflow attackis that, while it computes the Bounded Sum using integer arithmetic, the noise addition is done using floating-point arithmetic.

It is not clear whether there is an analogue of this solution for the use of sensitivity in the Exponential Mechanism (see Section 1.2.)

Changing Overflow Mode. As mentioned above, another solution for the case of k-bit integers is to replace wraparound with saturation arithmetic, combining it with either the random permutation technique or split summation in order to handle unordered adjacency relations.

Changing Rounding Mode. For our rounding attacks against floating-point numbers, another solution (beyond those based on sensitivity-via-accuracy) is to replace the default banker's rounding with another standard rounding mode, namely round toward zero (RTZ). We show that this gives an implemented sensitivity of

$$\Delta_{\simeq_{unbdd}} BS^{*}_{L,U,n} \leq \left(\max\left\{1 + O\left(\frac{n}{2^k}\right), 2\right\}\right) \cdot \max\{|U|, |L|\},$$

provided that (a) all elements of the datasets have the same sign (i.e., $L \geq 0$ or $U \leq 0$), and (b) we work with an ordered dataset adjacency relation. To handle the case of mixed signs, we can use the shifting technique (in case of bounded DP) or split summation (in the case of unbounded DP). To handle unordered dataset relations, we can apply the random permutation technique. Altogether these solutions maintain a sensitivity that is within a small constant factor of the idealized sensitivity in all cases.

6 ROADMAP FOR IMPLEMENTING SOLUTIONS

In this section, we present a set of recommendations aimed at DP practitioners who wish to fix the vulnerabilities that we have presented. Table 2 presents several solutions and their associated sensitivities. Several of the solutions can be implemented with only a few alterations to current code.

6.1 Integer Summation

We recommend the following solutions for integer summation:

(1) Modular sensitivity: In many programming languages, the default method for handling integer overflow is wraparound, which is equivalent to modular summation. Thus, many DP libraries luckily already implement the modular solution. One point of caution is that *both* the summation and noise addition steps must occur in this modular fashion. (Modular summation coupled with non-modular noise addition caused the privacy violation that we found in IBM's diffprivlib. Wraparound is a standard feature of arithmetic on integer data types, so this solution should not create new unexpected behaviors for data analysts.
If library maintainers are uncomfortable with the possibility of wraparound or unable to offer modular noise addition, we encourage the following two solutions.
(2) Checking parameters: In the bounded DP setting, perform a check on the parameters ensuring that overflow cannot occur, e.g., check that $n \cdot U < 2^k$ in the setting of unsigned k-bit integers.
(3) Split summation: In the unbounded DP setting, we recommend switching to saturation arithmetic (where overflow is handled by clamping to the range $[\min(T), \max(T)]$) and applying split summation, where we separately sum the positive and negative numbers. For L and U both non-negative or both non-positive, split summation is equivalent to standard summation. If split summation is not desirable, then we recommend using saturation arithmetic, and either applying a randomized permutation to the dataset or using an ordered notion of neighboring datasets and being careful about stability with respect to ordering in all dataset transformations.

6.2 Floating-point Summation

For libraries that have or are implementing a (correct) version of (noisy) integer summation, we recommend using our Float2Int solution if feasible.

If floating-point summation must be used (e.g. due to a machine-learning pipeline or exernal compute engine that has hardwired numerical types), we recommend using the implemented sensitivity bounds that come from accuracy bounds (see Table 2) for whatever floating-point summation method is already implemented. Typically this will be Iterated Summation, but some libraries (e.g., those based on `numpy`) may already using a better method like Pairwise Summation. For bounded DP, using these sensitivities only requires also checking the parameters to ensure that overflow cannot occur. To achieve unbounded DP, truncated summation together with a random permutation and should be used as well. These solutions should work very well for 64-bit floats, as they give an implemented sensitivity that is at most $1.5 \cdot U$ for datasets of size smaller than 67 million. If it is important to have sensitivity close to $U - L$, then the "shifting bounds" technique can be used as well.

For data consisting of lower-precision floats, another attractive approach is to keep the values as low-precision floats (e.g., to keep the memory footprint small in machine learning pipelines) but accumulate the sum in a 64-bit float, which will allow the above solutions to apply. For huge datasets (e.g., more than 67 million

Data type	Solution name		$\frac{\textit{implemented sensitivity}}{\textit{idealized sensitivity}}$	Conditions
ints	RP			
	Dataset adjacency relations			
	Checking parameters			
	Modular noise addition			
Floats	Reducing floats to ints		$1 + O(n\sqrt{n}/2^m)$	
Floats	RP + Split summation + RTZ		$\min\{(1 + O(n/2^k)), 2\}$	$\text{sign}(U) = \text{sign}(L)$
			$\min\{(2 + O(n/2^k)), 5\}$	$\text{sign}(U) \neq \text{sign}(L)$
Floats	Sensitivity from accuracy + Truncated summation	Iterative	$1 + \Theta(n^2/2^k)$	
		Pairwise	$1 + O(n \log(n)/2^k)$	$n < 2^k$
		Kahan	$1 + O(n/2^k)$	$n < 2^k$

Table 2: Upper bounds obtained in our solutions for numerical type T, $-U \leq L \leq U$, and datasets of length n. The parameter k is the number of bits (for k-bit ints) and the mantissa length (for (k, ℓ)-bit floats); the parameter m is the bit-length of the integer data type to which the floats were reduced. We remark that in the iterative, pairwise, and Kahan sensitivities, the 1 factor becomes a 2 in the case where n is unknown and $\text{sign}(U) \neq \text{sign}(L)$. Note that all the methods listed in a given solution box need to be used in combination (e.g., RP, Split Summation, and RTZ all need to be used together).

records), a 128-bit accumulator could be used or the summation method could be switched to a method where the accuracy bounds grow more slowly with n (e.g., pairwise or Kahan summation). Otherwise, we recommend considering the "changing rounding mode" solution, which requires changing the rounding mode from the standard banker's rounding to round toward zero.

ACKNOWLEDGMENTS

The authors would like to thank Grace Tian for her contributions in the preliminary stages of this work.

Sílvia Casacuberta was supported by the Harvard Program for Research in Science and Engineering (PRISE) and a grant from the Sloan Foundation. Michael Shoemate was supported by a grant from the Sloan Foundation, a grant from the US Census Bureau, and gifts from Apple and Facebook. Salil Vadhan was supported by a grant from the Sloan Foundation, gifts from Apple and Facebook, and a Simons Investigator Award. Connor Wagaman was supported by the Harvard College Research Program (HCRP); much of this work was completed while he was at Harvard University.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our funders.

REFERENCES

[1] Martín Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: ACM, 2016, pp. 308–318. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978318. URL: http://doi.acm.org/10.1145/2976749.2978318.

[2] John M. Abowd. "The U.S. Census Bureau Adopts Differential Privacy". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: ACM, 2018, pp. 2867–2867. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3226070. URL: https://digitalcommons.ilr.cornell.edu/ldi/49/.

[3] Victor Balcer and Salil Vadhan. "Differential Privacy on Finite Computers". In: *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Ed. by Anna R. Karlin. Vol. 94. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 43:1–43:21. ISBN: 978-3-95977-060-6. DOI: 10.4230/LIPIcs.ITCS.2018.43. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8353.

[4] Aleix Bassolas et al. "Hierarchical organization of urban mobility and its connection with city livability". In: *Nature communications* 10.1 (2019), pp. 1–10.

[5] Shailesh Bavadekar et al. "Google COVID-19 Search Trends Symptoms Dataset: Anonymization Process Description (version 1.0)". In: *arXiv preprint arXiv:2009.01265* (2020).

[6] Shailesh Bavadekar et al. "Google COVID-19 Vaccination Search Insights: Anonymization Process Description". In: *arXiv preprint arXiv:2107.01179* (2021).

[7] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. "Practical privacy: the SuLQ framework". In: *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM. 2005, pp. 128–138.

[8] Mark Bun and Thomas Steinke. "Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds". In: *CoRR* abs/1605.02065 (2016). URL: http://arxiv.org/abs/1605.02065.

[9] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. "The Discrete Gaussian for Differential Privacy". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/b53b3a3d6ab90ce0268229151c9bde11-Abstract.html.

[10] Sílvia Casacuberta, Michael Shoemate, Salil Vadhan, and Connor Wagaman. *Widespread Underestimation of Sensitivity in Differentially Private Libraries and How to Fix It*. 2022. DOI: 10.48550/arxiv.2207.10635. URL: https://arxiv.org/abs/2207.10635.

[11] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. "Collecting telemetry data privately". In: *Advances in Neural Information Processing Systems* 30 (2017).

[12] Jinshuo Dong, Aaron Roth, and Weijie J Su. "Gaussian differential privacy". In: *arXiv preprint arXiv:1905.02383* (2019).

[13] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. "Our data, ourselves: privacy via distributed noise generation". In: *Advances in cryptology—EUROCRYPT 2006*. Vol. 4004. Lecture Notes in Comput. Sci. Springer, Berlin, 2006, pp. 486–503. DOI: 10.1007/11761679_29. URL: http://dx.doi.org/10.1007/11761679_29.

[14] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. "Calibrating noise to sensitivity in private data analysis". In: *Journal of Privacy and Confidentiality* 7.3 (2016). To appear. Preliminary version in *Proc. TCC '06*.

[15] Cynthia Dwork and Guy N. Rothblum. "Concentrated Differential Privacy". In: *CoRR* abs/1603.01887 (2016). arXiv: 1603.01887. URL: http://arxiv.org/abs/1603.01887.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the*

2014 ACM SIGSAC Conference on Computer and Communications Security. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 1054–1067. ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660348. URL: http://doi.acm.org/10.1145/2660267.2660348.

[17] Andrew David Foote, Ashwin Machanavajjhala, and Kevin McKinney. "Releasing Earnings Distributions using Differential Privacy: Disclosure Avoidance System For Post-Secondary Employment Outcomes (PSEO)". In: *Journal of Privacy and Confidentiality* 9.2 (2019).

[18] Marco Gaboardi, Michael Hay, and Salil Vadhan. "A programming framework for OpenDP". In: *Manuscript* (2020).

[19] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. "Preserving differential privacy under finite-precision semantics". In: *Theoretical Computer Science* 655 (2016), pp. 92–108.

[20] A. Ghosh, T. Roughgarden, and M. Sundararajan. "Universally Utility-maximizing Privacy Mechanisms". In: *SIAM Journal on Computing* 41.6 (2012), pp. 1673–1693. DOI: 10.1137/09076828X. eprint: https://doi.org/10.1137/09076828X. URL: https://doi.org/10.1137/09076828X.

[21] *Google's differential privacy libraries*. GitHub repository. URL: https://github.com/google/differential-privacy.

[22] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. "Differential Privacy Under Fire". In: *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011. URL: http://static.usenix.org/events/sec11/tech/full%5C_papers/Haeberlen.pdf.

[23] A Herdağdelen, A Dow, S Bogdan, M Payman, and A Pompe. "Protecting privacy in Facebook mobility data during the COVID-19 response". In: *Facebook Research* (2020).

[24] Nicholas J. Higham. "The Accuracy of Floating Point Summation". In: *SIAM J. Sci. Comput.* 14.4 (1993), pp. 783–799. DOI: 10.1137/0914050. URL: https://doi.org/10.1137/0914050.

[25] Naoise Holohan and Stefano Braghin. "Secure Random Sampling in Differential Privacy". In: *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Vol. 12973. Lecture Notes in Computer Science. Springer, 2021, pp. 523–542. DOI: 10.1007/978-3-030-88428-4_26. URL: https://doi.org/10.1007/978-3-030-88428-4%5C_26.

[26] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. "Diffprivlib: The IBM Differential Privacy Library". In: *CoRR* abs/1907.02444 (2019). arXiv: 1907.02444. URL: http://arxiv.org/abs/1907.02444.

[27] Christina Ilvento. "Implementing the Exponential Mechanism with Base-2 Differential Privacy". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM, 2020, pp. 717–742. DOI: 10.1145/3372297.3417269. URL: https://doi.org/10.1145/3372297.3417269.

[28] Jiankai Jin, Eleanor McMurtry, Benjamin IP Rubinstein, and Olga Ohrimenko. "Are We There Yet? Timing and Floating-Point Attacks on Differential Privacy Systems". In: *arXiv preprint arXiv:2112.05307* (2021).

[29] Noah Johnson, Joseph P. Near, and Dawn Song. "Towards Practical Differential Privacy for SQL Queries". In: *Proc. VLDB Endow.* 11.5 (Jan. 2018), pp. 526–539. ISSN: 2150-8097. DOI: 10.1145/3187009.3177733. URL: https://doi.org/10.1145/3187009.3177733.

[30] Noah M. Johnson, Joseph P. Near, Joseph M. Hellerstein, and Dawn Song. "Chorus: a Programming Framework for Building Scalable Differential Privacy Mechanisms". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 535–551. DOI: 10.1109/EuroSP48549.2020.00041. URL: https://doi.org/10.1109/EuroSP48549.2020.00041.

[31] William Kahan. "Pracniques: further remarks on reducing truncation errors". In: *Communications of the ACM* 8.1 (1965), p. 40.

[32] Michael J. Kearns. "Efficient Noise-Tolerant Learning from Statistical Queries". In: *Journal of the Association for Computing Machinery* 45.6 (1998), pp. 983–1006. DOI: 10.1145/293347.293351. URL: http://doi.acm.org/10.1145/293347.293351.

[33] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks". In: *International conference on machine learning*. PMLR. 2016, pp. 2849–2858.

[34] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. "Privacy: Theory Meets Practice on the Map". In: *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. ICDE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 277–286. ISBN: 978-1-4244-1836-7. DOI: 10.1109/ICDE.2008.4497436. URL: https://doi.org/10.1109/ICDE.2008.4497436.

[35] Frank McSherry. "Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis". In: *Communications of the ACM* 53.9 (Sept. 2010), pp. 89–97. ISSN: 0001-0782. DOI: 10.1145/1810891.1810916. URL: http://doi.acm.org/10.1145/1810891.1810916.

[36] Solomon Messing et al. *Facebook Privacy-Protected Full URLs Data Set*. Version DRAFT VERSION. 2020. DOI: 10.7910/DVN/TDOAPG. URL: https://doi.org/10.7910/DVN/TDOAPG.

[37] Ilya Mironov. "On Significance of the Least Significant Bits for Differential Privacy". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 650–661. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382264. URL: http://doi.acm.org/10.1145/2382196.2382264.

[38] Ilya Mironov. "Rényi Differential Privacy". In: *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017, pp. 263–275. ISBN: 978-1-5386-3217-8. DOI: 10.1109/CSF.2017.11. URL: https://doi.org/10.1109/CSF.2017.11.

[39] Ilya Mironov, Kunal Talwar, and Li Zhang. "Rényi Differential Privacy of the Sampled Gaussian Mechanism". In: *arXiv preprint arXiv:1908.10530* (2019).

[40] Mayana Pereira et al. "US Broadband Coverage Data Set: A Differentially Private Data Release". In: *arXiv preprint arXiv:2103.14035* (2021).

[41] Ryan Rogers et al. "A Members First Approach to Enabling LinkedIn's Labor Market Insights at Scale". In: *arXiv preprint arXiv:2010.13981* (2020).

[42] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. "Airavat: Security and Privacy for MapReduce". In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California: USENIX Association, 2010, pp. 20–20. URL: http://dl.acm.org/citation.cfm?id=1855711.1855731.

[43] Jayshree Sarathy and Salil Vadhan. "Analyzing the Differentially Private Theil-Sen Estimator for Simple Linear Regression". In: *arXiv preprint arXiv:2207.13289* (2022).

[44] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and XiaoFeng Wang. "Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12". In: *CoRR* abs/1709.02753 (2017). arXiv: 1709.02753. URL: http://arxiv.org/abs/1709.02753.

[45] Shibo Wang and Pankaj Kanwar. *BFloat16: The secret to high performance on Cloud TPUs*. en. Aug. 2019. URL: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus/.

[46] James H Wilkinson. "Error analysis of floating-point computation". In: *Numerische Mathematik* 2.1 (1960), pp. 319–340.

[47] Royce J. Wilson et al. "Differentially Private SQL with Bounded User Contribution". In: *CoRR* abs/1909.01917 (2019). arXiv: 1909.01917. URL: http://arxiv.org/abs/1909.01917.

[48] Ashkan Yousefpour et al. "Opacus: User-Friendly Differential Privacy Library in PyTorch". In: *CoRR* abs/2109.12298 (2021). arXiv: 2109.12298. URL: https://arxiv.org/abs/2109.12298.