Are We There Yet? Timing and Floating-Point Attacks on Differential Privacy Systems

Jiankai Jin*, Eleanor McMurtry†‡, Benjamin I. P. Rubinstein*, Olga Ohrimenko*
*School of Computing and Information Systems, The University of Melbourne
†Department of Computer Science, ETH Zurich

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833672

***Abstract*—Differential privacy is a de facto privacy framework that has seen adoption in practice via a number of mature software platforms. Implementation of differentially private (DP) mechanisms has to be done carefully to ensure end-to-end security guarantees. In this paper we study two implementation flaws in the noise generation commonly used in DP systems. First we examine the Gaussian mechanism's susceptibility to a floating-point representation attack. The premise of this first vulnerability is similar to the one carried out by Mironov in 2011 against the Laplace mechanism. Our experiments show the attack's success against DP algorithms, including deep learning models trained using differentially-private stochastic gradient descent.**

In the second part of the paper we study discrete counterparts of the Laplace and Gaussian mechanisms that were previously proposed to alleviate the shortcomings of floating-point representation of real numbers. We show that such implementations unfortunately suffer from another side channel: a novel timing attack. An observer that can measure the time to draw (discrete) Laplace or Gaussian noise can predict the noise magnitude, which can then be used to recover sensitive attributes. This attack invalidates differential privacy guarantees of systems implementing such mechanisms.

We demonstrate that several commonly used, state-of-the-art implementations of differential privacy are susceptible to these attacks. We report success rates up to 92.56% for floating point attacks on DP-SGD, and up to 99.65% for end-to-end timing attacks on private sum protected with discrete Laplace. Finally, we evaluate and suggest partial mitigations.

I. Introduction

Given the equation

$$z - y = 0.1234567890004\ ,$$

can y be equal to 0, 2000 *or* 20000*? Though one may ask "what is z?", it is possible to answer this question without knowing z, if one knows that the arithmetic was computed on a machine using the double-precision floating-point format. While $z = 2000.1234567890004$ cannot be represented,* 2000.1234567890003 *and* 2000.1234567890006 *can be. Similarly for $z = 20000.1234567890004$. In fact without knowing z at all, we can say definitively that y must equal* 0 *if it is known to be one of* 0, 2000, *or* 20000.

Differential privacy (DP) is a de facto privacy framework that has received significant interest from the research community and has been deployed by the U.S. Census Bureau, Apple, Google, Microsoft, and many others. Research on DP ranges from algorithms with different performance trade-offs, to new models in different settings, and also to practical implementations [1], [2], [3], [4], [5]. Robust implementations are crucial to provide end-to-end privacy that matches on-paper differential privacy guarantees.

‡Work done in part while at The University of Melbourne.

Implementations of DP algorithms often raise concerns not considered in theoretical analysis (which focuses on idealized settings). Mironov [6] was first to discuss the implications of the fact that one cannot represent—and thus cannot sample from—all real numbers on a finite-precision computer. Focusing on the Laplace mechanism, Mironov's attack proceeds by observing that certain floating-point values cannot be generated by a DP computation and hence a release could reveal the (private) noiseless value. On the other hand, Haeberlen *et al.* [7] and Andrysco *et al.* [8] showed that DP algorithms may suffer from timing side-channels since such algorithms can take different time depending on sensitive values in a dataset.

In this paper we extend Mironov's attack to other DP mechanisms and study its effects on real-world DP implementations. We then describe another timing side-channel that can arise in DP implementations due to the timing of the noise samplers.

a) Floating-Point Representation of the Gaussian Distribution: The *Gaussian mechanism* (based on additive Gaussian noise) is another well-studied DP mechanism. It achieves what is often called *approximate differential privacy*, meaning that the mechanism may fail completely to provide pure DP with some small and controllable probability δ. However, the mechanism has advantages over the Laplace mechanism, including lighter tails than the Laplace distribution and superior composition properties when answering many queries with independent noise. Generalizations like Rényi differential privacy [9] can perform tight composition analysis. Recently *truncated concentrated differential privacy* [10] has emerged as a promising generalization that bounds the residual privacy loss from approximate DP, allows efficiently-computable optimal composition, and captures privacy amplification by subsampling as present in [11]. Because of these advantages, the Gaussian mechanism is often the tool of choice for applications such as deep learning [11] that involve carefully controlled privacy budgets over sequences of releases.

A question arises: are the same attacks as in [6] possible against the Gaussian mechanism? Though several works [12], [13] mention that it may be feasible, to the knowledge of the authors no one has demonstrated this possibility nor shown how to carry out this attack in practice. In this paper we study these two questions and demonstrate an attack confirming that common implementations of Gaussian sampling are subject to floating-point attacks.

Unfortunately directly using the attack from [6] is not possible since Gaussian noise is drawn using very different techniques to Laplace. The main challenge is due to some Gaussian samplers being based on two random values and not one. Such methods produce two independent Gaussian samples, and most implementations cache one of them to be used next time the mechanism is called. We develop an attack that uses both of these values. Due to the two equations with several unknowns that the attacker needs to solve, there can be more than one pair of feasible values. As a result our attack can yield false positives. Nevertheless, we show that the attack is still feasible and has a significant success rate. Additionally, we show that a Gaussian sampler based on a different method that generates only one sample is also susceptible to a floating-point attack, and the attack succeeds at a higher rate than for samplers based on two values.

The most prominent recent use of the Gaussian mechanism is in training machine learning (ML) algorithms using *differentially-private stochastic gradient descent* (DP-SGD) [11]. We show that we can mount the attack in this setting to determine if a batch contains a particular record or not, violating DP guarantees of DP-SGD. Moreover, ML model training naturally reveals sequential Gaussian samples to an adversary, as it returns a noisy gradient for each parameter of the model.

b) Timing Attacks Against Discrete Distribution Samplers: In the second part of the paper we study the primary method that has been proposed to defend against floating-point attacks: discrete versions of the Laplace and Gaussian mechanisms [12], [13]. These approaches employ sampling algorithms that make no use of floating-point representations. We observe that such mechanisms, though defending against floating-point attacks, are susceptible to a timing side channel: an adversary who observes the time it takes to draw a sample can determine the generated noise's magnitude. When used within a DP mechanism, our attack reveals the noise contained in the result, and thus reveals the noiseless (private) value.

Our timing attack is possible due to the underlying technique that these discrete samplers rely on: direct simulation of *geometric sampling*, meaning values are sampled until a coin toss results in a "head". The number of such coin tosses is tied to the magnitude of the noise returned; timing the sampler reveals this number and thus leaks the noise magnitude. Though timing has been identified as a potential side-channel in DP [7], [8], to our knowledge we are the first to show that noise distribution samplers and not the mechanisms themselves give rise to secret-dependent runtimes.

Our contributions are:

- We show that the Gaussian mechanism of differential privacy suffers from a side channel due to floating-point representation. To this end, we devise attack methods to show how to exploit this vulnerability since the known floating-point attack against the Laplace mechanism cannot be used directly.
- We use the above results to demonstrate empirically that Gaussian samplers as implemented in `NumPy`, `PyTorch` and `Go` are vulnerable to our attack. Focusing on the Opacus DP library implementation [4] by Facebook, we also show that DP-SGD is vulnerable to information leakage under our attack. [*]
- We then observe that discrete methods developed to protect against floating-point attacks for both the Laplace and Gaussian mechanisms suffer from timing side channels. We show that two libraries are vulnerable to these attacks: a DP library by Google [3] and the implementation accompanying another work on discrete distributions in [12], [14].
- We discuss and evaluate mitigations against each attack.

Disclosure: We have informed maintainers of the DP libraries mentioned above of the results of this paper. They have acknowledged our report and notification of the disclosure dates.

II. Background

In this paper we develop attacks on differential privacy based on floating-point representation and timing channels. In this section, we give background on how floating-point values are represented on modern computers, differential privacy, and the Laplace and Gaussian mechanisms for DP.

A. Floating-Point Representation

Floating-point values represent real values using three numbers: a sign bit b, an exponent e, and a significand $d_1 d_2 \dots d_{\mathsf{d}}$. For example, 64-bit (double precision) floating-point numbers allocate 1 bit for b, 11 bits for e, and 52 bits for the significand. Such a floating-point number is defined to be $(-1)^{\mathsf{b}} \times (1.d_1 d_2 \dots d_{\mathsf{d}})_2 \times 2^{\mathsf{e}-1023}$.

Crucially, the number of real values representable using floating-point values in the ranges $[a_1, b_1]$ and $[a_2, b_2]$, $a_i < b_i$, are different even if $b_1 - a_1 = b_2 - a_2$. For example, there are approximately 2^{17} floating-point values in the range $[10, 10 + 2^{-32}]$, and there are approximately 2^{14} floating-point values in $[100, 100 + 2^{-32}]$. Thus, floating-point values are more densely distributed around 0.

B. Differential Privacy

Consider a collection of datasets $\mathcal{D}$ and an arbitrary space of output responses $\mathcal{R}$. We say that two datasets D, D' are *neighboring* if they differ on one record. A randomized mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ is (ϵ, δ)-*differentially private* [15] for $\epsilon > 0$ and $\delta \in [0, 1)$ if given any two neighboring datasets $D, D' \in \mathcal{D}$ and any subset of outputs $R \subseteq \mathcal{R}$ it holds that $\Pr[\mathcal{M}(D) \in \mathcal{R}] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in \mathcal{R}] + \delta$. That is, the probability of observing the same output y from $\mathcal{M}(D)$ and $\mathcal{M}(D')$ is bounded. DP therefore guarantees that given an output y, an attacker cannot determine which of D or D' was used for the input. If there is some output that is possible with D but not D', this inequality cannot hold for non-trivial δ, so

[*]Since notifying Facebook about the floating-point attack, Opacus library now has proposed a mitigation in https://github.com/pytorch/opacus/pull/260

the mechanism cannot be DP. This is the key fact our attacks exploit.

In this paper, we consider DP mechanisms $\mathcal{M}$ that provide protection by computing the intended function f on the data D and randomizing its output. That is, $\mathcal{M}(D) = f(D) + s$ where $s \leftarrow \mathsf{NoiseDist}$ is noise drawn either from Laplace or Gaussian distributions with appropriate parameters (as discussed below). For example, f may compute the sum of the set of employee incomes D, and we may wish to keep the exact values of the incomes private. Our attacks are based on observations about $\mathsf{NoiseDist}$ that can be used to learn the noise sampled from it.

C. Laplace Mechanism

The Laplace mechanism provides ϵ-DP by additive noise drawn from the Laplace distribution $\mathsf{Lap}(\lambda)$ (i.e., $\mathsf{NoiseDist}$ is $\mathsf{Lap}(\lambda)$). In the scalar-valued case $\mathcal{M}(D) = f(D) + \mathsf{Lap}(\lambda)$. Here we use a scale parameter $\lambda = \Delta/\epsilon$ where Δ is f's sensitivity, meaning any neighboring data sets D and D' satisfy $|f(D) - f(D')| \leq \Delta$.

D. Gaussian Mechanism

The Gaussian mechanism [16] provides (ϵ, δ)-DP to the outputs of a target function $f : \mathcal{D} \to \mathcal{R}$, where $\mathcal{R} = \mathbb{R}^d$. The mechanism is popular due to its favorable noise tails compared to alternative mechanisms, and its composition properties when the mechanism is used to answer many queries on data, as is common when training a machine learning model or answering repeated queries on a database [17], [18].

Let Δ_f be the L_2-*sensitivity* of f, that is, the maximum distance $\|f(D) - f(D')\|_2$ between any neighboring datasets D and D'. Then the Gaussian mechanism $\mathcal{M}(D)$ adds noise $\mathsf{NoiseDist}$ to $f(D)$. We write $\mathcal{N}(x, \sigma^2\Delta_f^2\mathbb{I})$ to mean the multivariate Gaussian with mean given by the target x and covariance given by the identity matrix scaled by $\sigma^2\Delta_f^2$. We then have $\mathsf{NoiseDist} = \mathcal{N}(0, \sigma^2\Delta_f^2\mathbb{I})$, and the output distribution is $\mathcal{N}(f(D), \sigma^2\Delta_f^2\mathbb{I})$. The resulting mechanism is (ϵ, δ)-DP if $\sigma = \sqrt{2\log(1.25/\delta)}/\epsilon$ for arbitrary $\epsilon > 0$ and $\delta \in (0, 1)$.

In addition to applications computing a one-off release of a function output, the Gaussian mechanism is commonly used repeatedly in training machine learning models using mini-batch stochastic gradient descent (SGD). This composite mechanism is called DP-SGD [19], [20]. When used to replace non-private mini-batch SGD, it produces a machine learning model with differential privacy guarantees on sensitive training data. This mechanism has been applied in Bayesian inference [21], to train deep learning models [11], [22], [23], and also in logistic regression models [20]. At a high level, a record-level DP-SGD mechanism aims to protect presence of a record in a batch and, hence, in the dataset. DP-SGD has been also used in the Federated Learning setting [22] where each client computes a gradient on their local data batch, adds noise and sends the result to a central server.

III. Threat Model

We consider an adversary that obtains an output of an implementation of a differentially-private mechanism, for example a DP-protected average income of people in a database of personal records, or DP-protected gradients used to train a machine learning model. DP aims to defend information about presence (or absence) of a certain record by providing plausible deniability. We show that the attacker can use artefacts of implementations of noise samplers in DP mechanisms to undermine their guarantees. In particular we consider two attacks based on separate artefacts — one on floating-point representation and one on timing.

We envision three scenarios where our attacks can be carried out:

- S: a member of the public observes *statistics* computed on sensitive data and protected with DP noise (e.g., those released by the US Census Bureau [24]);
- DB: an analyst interactively queries a differentially private *database* [25], [26], [7] which allows them to ask several queries of datasets and which transparently adds noise to preserve privacy of the dataset from the analyst;
- FL: a central server who is coordinating *federated learning* by collecting gradients from clients [22]. Here, a client adds noise to a gradient computed on its data to protect its data from the central server.

For all three scenarios above, our threat model builds on the threat model of DP where (1) the attacker observes a DP-protected output, (2) the attacker may know all the other records in the dataset except for the one record it is trying to guess, and (3) knows how the mechanism is implemented [†], but does not know the randomness used by it.

We describe additional adversarial capabilities required for each scenario and attack below.

a) Floating-Point Attack (Section IV-C): For one of our two floating-point attacks, in addition to observing a single DP output that the adversary wishes to attack, we assume the adversary has access to an output of a consecutive execution of a DP mechanism or its noise sampling. This is achievable in practice in all scenarios above due to:

1) multiple queries: In scenario S multiple statistics are released, in scenario DB a (malicious) analyst could query a DP protected database several times.
2) d-dimensional query: in all three scenarios, an output being protected can correspond to an output of a d-dimensional function where independent noise is added to each component such as a histogram [16] or a gradient computed for multiple parameters of ML model [11]. Moreover, gradients are assumed to be revealed as part of the privacy analysis in the central setting as well as in the FL setting.

b) Timing Attack (Section VI): In contrast to the floating-point attack above, here the threat model assumes that the attacker observes a DP output *and* can measure the time it takes for the DP mechanism to compute it. The attacker must also be able to measure multiple runs of an algorithm in order to obtain a baseline of running times. However during the

[†]Many DP implementations are open-source including [2], [3], [4], [5].

attack itself, the attacker only needs to make one observation to make a reasonable guess.

The attack can be deployed in the three scenarios above if an attacker has black-box access to the machine running DP code, similarly to the threat model of other timing side-channels used against DP mechanisms that are *not* based on noise samplers [8], [7] (see Section IX for more details). For example, the attacker may share a machine based in the cloud [1], [27], [28] or is a cloud provider itself. For the DB scenario specifically, a malicious analyst querying the mechanism hosted locally can readily measure the time it takes for the query to return. For the FL scenario (and remotely hosted databases in the DB scenario) the uncertainty in measuring precise time due to network communication can be reduced with recent attacks exploiting concurrent requests [29].

IV. Floating-Point Attack on Normal Distribution Implementations

We describe the floating-point attack that aims to determine whether a given floating-point value could have been generated by an instance of a Gaussian distribution or not. If not, this eliminates the possibility that a DP mechanism could have used this noise, hence undermining its privacy guarantees. We begin with a description of a generic floating-point attack against DP and then describe two common implementations of Gaussian samplers—polar and Ziggurat—and how they can be attacked.

A. Floating-Point Attack on DP

The DP threat model assumes that the adversary knows neighboring datasets D, D' and function f. Given an output y of a DP mechanism, where either $y = f(D) + s$ or $y = f(D') + s'$ and $s, s' \leftarrow \mathsf{NoiseDist}$, the attacker's goal is to determine if D or D' was used in the computation of y.

Mironov [6] showed that due to an artefact in the implementation of NoiseDist for Laplace, some values of s are impossible. Hence given y, if the adversary knows that s is impossible then it must be the case that D' was used to compute y (and similarly for s'). This directly breaks the guarantee of DP which states that there is a non-zero probability for each of the inputs producing the observed output. We will show that mechanisms that use Gaussian noise for NoiseDist — whose implementation is more complicated than Laplace — are also susceptible to implementation artefacts.

In the rest of this section we develop a function $\mathsf{IsFeasibleNormal}(s)$ which returns true if a given noise value s could have been drawn from implementations of Gaussian distributions and false otherwise. The attacker then runs $\mathsf{IsFeasibleNormal}(s)$ and $\mathsf{IsFeasibleNormal}(s')$. If only one of them returns true, the attacker determines that the corresponding dataset was used in the computation. Otherwise, it makes a random guess.

B. Warmup: Feasible Random Floating Points

We describe how random double-precision floating-point values ("doubles") are sampled on modern computers using a function RandomFP and show that given a double x, one can determine if it was generated using RandomFP or not. This will serve as a warm-up for our attack against the Gaussian distribution over doubles.

Random real values in the range $(0, 1)$ can be drawn by choosing a random integer u from $[1, R)$ and then dividing it by the resolution $R = 2^p$, where the value of p varies by system. We abstract this process using a function RandomFP that chooses an integer u at random from $[1, R)$ and returns u/R. Given a double x one can determine if it could have been produced by RandomFP by checking if $x \overset{?}{=} \bar{u}/R$ for some integer $\bar{u} \in [1, R)$. If the equality holds then x could have been produced from a random integer. Since rounding errors are introduced during multiplication and division, we will later also perform the above check for neighboring values of x.

C. Polar Method: Implementation and Attack

In this section we describe the *polar method* and the floating-point attack against it.

1) Method: The *Marsaglia polar method* [30] is a computational method that generates samples of the standard normal distribution from uniformly distributed random values. PolarMethod operates as follows:

P1 Choose independent uniform random values x_1' and x_2' from $(0, 1)$ using RandomFP.
P2 Set $x_1 \leftarrow 2x_1' - 1$ and $x_2 \leftarrow 2x_2' - 1$. (Note that both fall in the interval $(-1, 1)$.)
P3 Set $r \leftarrow x_1^2 + x_2^2$.
P4 Repeat from Step **P1** until $r \le 1$ and $r \ne 0$.
P5 Set $s_1 \leftarrow x_1\sqrt{\frac{-2\log r}{r}}$ and $s_2 \leftarrow x_2\sqrt{\frac{-2\log r}{r}}$.
P6 Return s_1.

The procedure generates two independent samples from a normal distribution: s_1 and s_2. The second value, s_2, is cached and returned on the next invocation. If the cache is empty, the sampling method is invoked again. The method can generate samples from $\mathcal{N}(0, \sigma^2)$ by returning σs_1 and σs_2 instead.

The polar method is used by both the GNU C++ Library with `std::normal_distribution` and the Java class `java.util.Random` (in the `nextGaussian` method). A related technique called *Box-Muller method* described in the Appendix A that relies on computing sin and cos is used in PyTorch and was implemented in the older versions of Diffprivlib [5].

2) Floating-Point Attack: The attacker's goal is to devise a function $\mathsf{IsFeasibleNormal}(s)$ that determines if value s could have been generated by PolarMethod — a computational method for drawing normal noise. Here, we assume that the attacker knows sample s_2 and is trying to guess if $s \overset{?}{=} s_1$. As discussed in Section III, an attacker can learn s_2 either through multiple queries or multi-dimensional queries. We note that the attack against the Laplace method by Mironov [6] cannot be applied since PolarMethod (and the Box-Muller method) (1) relies on different mathematical formulae and (2) uses two random values and draw two samples from their target distribution. To this end, we devise an FP attack specifically for normal distribution implementations. We describe the attack for the polar method below. The attack for the Box-Muller method proceeds similarly, using trigonometric functions instead.

Before proceeding with the attack, we observe that r in PolarMethod can be expressed using s_1 and s_2 by simple arithmetic rearrangements based on steps **P3** and **P5**.

$$r = x_1^2 + x_2^2 = \left(s_1\sqrt{\frac{r}{-2\log r}}\right)^2 + \left(s_2\sqrt{\frac{r}{-2\log r}}\right)^2$$
$$= \frac{s_1^2 r}{-2\log(r)} + \frac{s_2^2 r}{-2\log(r)}$$

Rearranging this equation further, we obtain

$$-2\log r = s_1^2 + s_2^2$$
$$r = \exp\left(-\frac{s_1^2 + s_2^2}{2}\right) \tag{1}$$

The intuition behind our attack is similar to that of "attacking" RandomFP in Section IV-B: we find an expression that must be an integer if PolarMethod was used. We observe that $r \times R^2$ must be an integer: since x_1' and x_2' are produced using RandomFP (step **P1**) there must be integers u_1 and u_2 such that $x_1' = u_1/R$ and $x_2' = u_2/R$. Rearranging and substituting these x_1' and x_2' further in steps **P2** and **P3** we obtain

$$r = (2u_1/R - 1)^2 + (2u_2/R - 1)^2$$

Since u_1, u_2, R are integers, value $r \times R^2$ must be an integer.

IsFeasibleNormal(s) proceeds as follows. The attacker computes value of $r \times R^2$ using Equation 1 with values s (instead of s_1) and s_2 as follows:

$$\exp\left(-\frac{s^2 + s_2^2}{2}\right) \times R^2 \tag{2}$$

It then checks if the value in Equation 2 is an integer. If it is, then IsFeasibleNormal(s) returns true since s and s_2 could be produced using PolarMethod.

As floating-point arithmetic cannot be done with infinite precision on finite machines, s and s_2 might be inaccurate. To this end, we perform a heuristic search in each direction of s and s_2 where we try several neighboring values of s and s_2. In our experiments, we choose to search 50 values in each direction. This search heuristic is prone to errors and may result in false positives and false negatives, due to (1) more than one pair of values resulting in s_1 and s_2 and (2) our search not being exhaustive in exploring values with a range of 100 values. Nevertheless in the next section we show that the attack is still successful for a variety of applications of Gaussian noise in DP.

D. Ziggurat Method: Implementation and Attack

The Ziggurat method [31] is another method for generating samples from the normal distribution. Compared to the Box-Muller and polar methods, it generates one sample on each invocation. It is a rejection sampling method that randomly generates a point in a distribution slightly larger than the Gaussian distribution. It then tests whether the generated point is inside the Gaussian distribution.

1) Method: ZigguratMethod relies on three precomputed tables $w[n]$, $f[n]$ and $k[n]$ that are directly stored in the source code. The Ziggurat implementation in Go uses $n = 128$ and proceeds as follows.

Z1 Generate a random 32-bit integer j and let i be the index provided by the rightmost 7 bits of j.
Z2 Set $s \leftarrow jw[i]$. If $j < k[i]$, return s.
Z3 If $i = 0$, run a fallback algorithm for generating a sample from the tails of the distribution.
Z4 Use RandomFP to generate independent uniform random value U from $(0, 1)$. Return s if:

$$U\left(f[i-1] - f[i]\right) < f(s) - f[i]$$

Z5 Repeat from step **Z1**.

We refer interested readers to [32] for how to sample from the tail of a normal distribution and to [31] for how the tables $w[n], f[n], k[n]$ are generated. The method can generate samples from $\mathcal{N}(0, \sigma^2)$ by returning σs.

The Ziggurat method is used by the Go package `math/rand` with `NormFloat64` and new random sampling methods of `NumPy`‡.

2) Attack: This time, the attacker aims to devise a function IsFeasibleNormal(s) that determines if value s could have been generated by ZigguratMethod. We describe our attack steps below since attacks in Section IV-C and [6] are not applicable due to pre-computed tables used in Ziggurat.

We observe that the returned value is $jw[i]$ in steps **Z2** and **Z4**, except when the value is sampled from the tail of Gaussian distribution in step **Z3** (which happens infrequently). Furthermore, j is an integer and all values of $w[n]$ are available to the attacker, because all precomputed tables are stored directly in the source code. Based on these observations, for a noise s, our attack for ZigguratMethod proceeds as:

1) For each $w[i]$, $i \in [1, n]$ calculate $\mathsf{w}[i] = s/(\sigma w[i])$.
2) For each $\mathsf{w}[i]$, check if it is an integer.
3) If any $\mathsf{w}[i]$ is an integer, then we take s as a feasible floating-point value, IsFeasibleNormal(s) returns true.

Note that the method does not attack values sampled from the tail. However, we observed that it happens less than 0.1% of the time when $n = 128$, hence, the attack works for the majority of the cases.

V. Experiments: FP Attack on Normal Distribution

We perform four sets of experiments to evaluate leakage of Gaussian samplers based on the attack described in the previous section. First, we enumerate the number of possible values in a range of floating points that can result in an attack and observe that it is not negligible. We then show the effectiveness of our attack on private count and DP-SGD, a popular method designed for training machine learning (ML) models under differential privacy [11].

‡https://numpy.org/doc/stable/reference/random/index.html

Gaussian Implementations: In our experiments, we use four implementations of Gaussian distribution samplers.

- Gauss_polar: our implementation of polar method as described in Section IV-C1.
- Gauss_numpy: `NumPy` implementation of polar Gaussian sampling.
- Gauss_pytorch: `PyTorch` implementation of Box-Muller Gaussian sampling.
- Gauss_go: `Go` implementation of Ziggurat Gaussian sampling.

Our attacks on DP-SGD were tested using the privacy engine implementation of the Opacus library [4]*.

A. Distribution of "Attackable" Values

In this section we aim to understand how many floating-point values could not have been generated from a Gaussian sampling implementation. That is, for how many values s, IsFeasibleNormal(s) would return false. We call such values "attackable" since if an adversary were to observe them, they would know that the value must have been produced in a specific manner (e.g., by adding noise to $f(D')$ as opposed to $f(D)$). We use the Gauss_polar implementation to perform this experiment in a controlled environment and assume that the adversary is given y_1 and y_2 where $y_1 = f(D) + s_1$ and $y_2 = f(D) + s_2$ or $y_1 = f(D') + s'_1$ and $y_2 = f(D') + s'_2$. (Compared to the attack in Section IV the attacker is not given s_2 directly, however the attack can proceed similarly.)

For each pair of $y_1 = f(D) + s_1$ and $y_2 = f(D) + s_2$, if the attack successfully concludes that they are in support of $f(D)$, and not $f(D')$, we count them as attackable values. We set $f(D) = 0$, $f(D') = 1$, $\sigma = 114$, $R = 2^{10}$. The blue line of Figure 1 shows the distribution of the rate of attackable values where we plot y_1 and y_2. In order to fit all measured floating-points into the resolution of the graph, we aggregate the results over small intervals of width 0.8.

Mironov generated a similar graph for the Laplace distribution in [6]. Compared to Gaussian, the Laplace distribution has more attackable values since it uses only one random value, hence the attack does not suffer from false negatives.

In order to understand our results further we also plot the average number of times each $y = f(D)+s$ is observed, using the gray line of Figure 1. This explains some of the spikes in the blue line: more frequently observed values indicate more attackable values. Percentage of floating-points that are attackable are higher closer to 0 (recall that the graph shows an average over FP intervals). Overall, the graph suggests that attackable values do exist and as we show in the rest of this section provide an avenue for an attack against DP mechanisms.

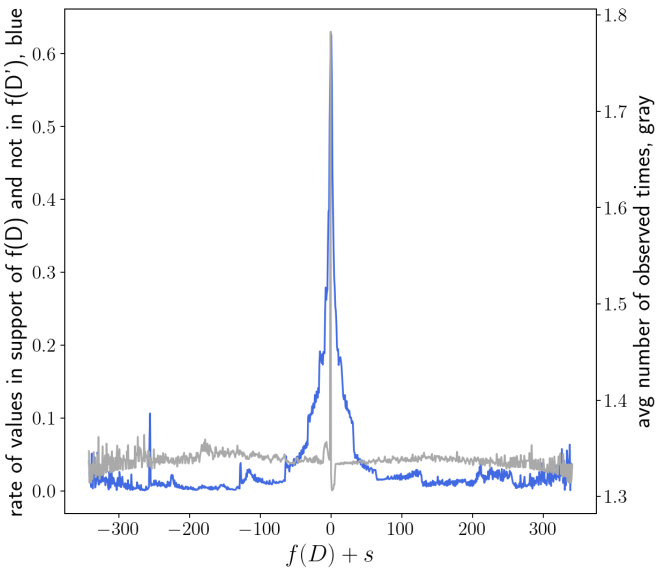


Fig. 1: Distribution and number of values amenable to a floating-point attack against a Gaussian implementation Gauss_polar using $R = 2^{10}$ and $\sigma = 114$. Here the horizontal axis, $f(D)+s$, shows both values $f(D)+s_1$ and $f(D)+s_2$. For presentation purposes, rate of attackable floating-point values is averaged over 0.8-wide intervals. Blue line indicates the rate of floating-point values that are in the support of $f(D) = 0$ and not in $f(D') = 1$. Gray line measures average number of times support for $f(D)$ was observed.

B. DP Gaussian Mechanism

1) Private count: In this section, we explore the effectiveness of our attack against the Gaussian mechanism used to protect a private count (i.e., corresponding to S or DB scenarios in Section III). We use the German Credit Dataset [33] and the query: count the number of records with number of credits greater than 16K (resulting in one record since the two highest values among dataset's 1000 records are 15945 and 18424).

Given the output of the Gaussian mechanism, y, the adversary's goal is to determine whether noiseless output came from $q = f(D)$ or $q' = f(D')$ where f is the count query defined above. The private count of values in a dataset is computed as: $y = q + s$ or $y = q' + s$, where q and q' are the outputs of f on dataset D and D', respectively, and s is a noise sampled with Gauss_numpy, Gauss_pytorch, or Gauss_go. For Gauss_numpy and Gauss_pytorch, the adversary also knows the second value sampled from the distribution (i.e., s_2 in Section IV-C). The neighboring datasets D and D' that our attack tries to distinguish differ on a single record whose credits is 0 in D and 18424 in D', so the count of records satisfying the above query is $q = 0$ for D and $q' = 1$ for D'.

We set the sensitivity $\Delta = 1$ since one record can change the output of f only by 1. We vary ϵ in the range $(0, 100]$ with fixed $\delta = 10^{-5}$. For each tuple of ϵ, δ and Δ, we use the analytic Gaussian Mechanism [34] to calculate the required noise scale σ for the experiment. Here, small ϵ and δ model a common set of parameters used in DP [35].

In Figure 2 we plot our success rate from 1 million trials from the following attack. Recall that the attacker cannot always make a guess due to the limitations listed in Section V-A.

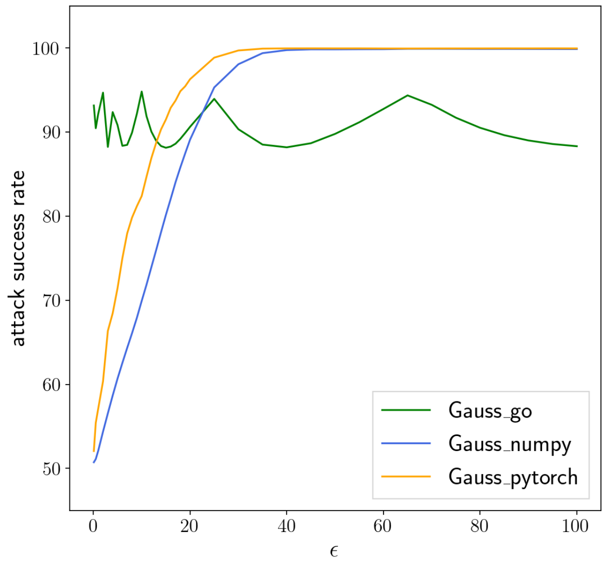


Fig. 2: FP attack success rate on private count where the count is protected with one of the three Gaussian samplers across $\epsilon \in (0, 100]$ with fixed $\delta = 10^{-5}$ and function sensitivity $\Delta = 1$. Baseline (random) attack success is 50%.

To this end, if the attack can find support for either only q or only q', then the attack outputs the corresponding guess. If the result is in support of both or neither of q and q', then the attacker is unsure and resorts to the baseline attack, choosing q or q' at random.

We observe that the attack is more successful for the Gauss_go method for values of ϵ less than 10. This is also the range of values for ϵ used in the literature on DP [36]. We note a slight cyclic behavior and observe that peaks occur when the corresponding noise scale σ (w.r.t. ϵ) is a power of 2 (see Figure 7 in the Appendix). We hypothesize that the success rate at those points is higher since multiplication and division by powers of 2 can be done via bit shifting that decreases the impact of rounding. Hence, extraction of s is not affected by rounding errors that would otherwise be introduced during the multiplication of s by σ in the ZigguratMethod and step 1 of the corresponding attack in Section IV-D.

Among two sampler methods, attack is more efficient against Gauss_pytorch than Gauss_numpy. We also observe that the attack becomes stronger as ϵ increases. This is potentially due to the higher distribution of attackable values as indicated in Figure 1. The attack is always more successful than the baseline random attack.

In Appendix D we also investigate the rate at which the attacker can make a guess (attack rate) and how many of these guesses are correct (attack accuracy) across a range of ϵ and sensitivity values. Gauss_numpy and Gauss_pytorch have attack rates ranging between 1.7% and 92.8% with attack accuracy of at least 89%. In comparison, the attack rates and accuracy against Gauss_go are at least 76% and 99%, respectively.

C. DP-SGD in Federated Learning scenario

The Gaussian mechanism is used extensively in training ML models via differentially-private stochastic gradient descent [11], [23], [21]. In this section, we describe successful attacks on the differentially-private training of ML models. Here, we consider the Federated Learning scenario from Section III where an attacker (central server) observes a DP-protected gradient computed on a batch of client's data and is trying to determine if the batch includes a particular record or not.

The main observations we make in this section are:

- An adversary can determine if a batch contains a record with a different label from other records in the batch, i.e., a record that comes from the same distribution as training dataset but different to those in the batch.
- The success rate of the attack increases as ϵ decreases as there are fewer floating-point values to represent large noise values.

1) Setup: We use Opacus library [4] for training machine learning models with DP-SGD. Our experiments are based on the Opacus example on MNIST data. This dataset [37] contains 70K images, split in 60K and 10K sets for training and testing, respectively. Each image is in 28×28 gray-scale representing a handwritten digit ranging from 0 to 9 where the written digit is the label of the image. The ML task is: given an unlabelled image, predict its label.

We use the same parameters for preprocessing and neural network training as used in the library [38], [4], following [11]. These training parameters are: learning rate 0.1, epoch number 8, batch size $S = 64$, number of batches in each epoch 100, and DP-SGD clipping norm $L = 1$. We vary ϵ to understand how different parameters affect the attack success rate while keeping a fixed $\delta = 10^{-5}$ (as in [38], [11]). Depending on the ϵ, the noise is drawn from a normal distribution with $\mu = 0$ and $\sigma \in [1, 250]$. As a result ϵ ranges from 0.1 to 0.68. Since the model has $d = 26,010$ parameters, each batch is used in d gradient computations to update the corresponding parameters. Hence, the attacker obtains d floating-point values protected with DP-SGD. Details of DP-SGD computation are provided in Appendix B.

We consider a setting where an adversary observes a gradient computed on FL client's batch of labelled MNIST images. The batch could correspond either to records B or a neighboring batch B' produced by replacing a randomly chosen record in batch B with a *canary record* (defined below). The attacker's goal is, given B, B' and a noisy gradient protected with Gaussian noise, to determine if the gradient was computed on B or B'.

We use different types of batches and canary records to test the effectiveness of our attack:

- SimLabelCanary: batch B is composed of shuffled training data. The canary record of its neighboring batch B' is a record drawn from the test dataset.

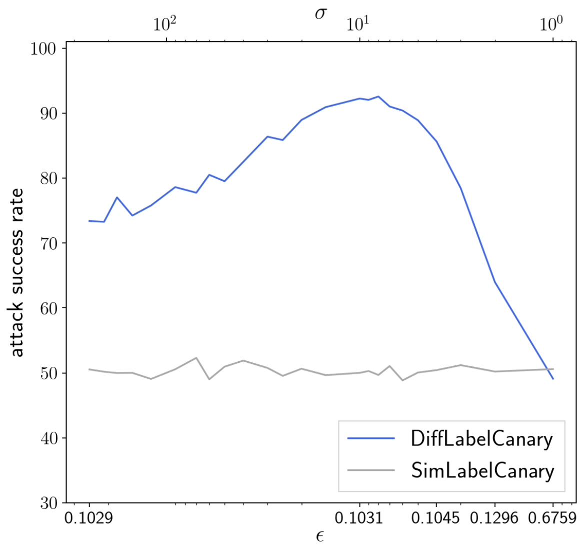


Fig. 3: FP attack success rate against DP-SGD on MNIST model using the Opacus [4]. DiffLabelCanary: attacker is distinguishing between a batch where all records have identical labels in the range $[1, 9]$ and a batch that has same records as well as an image for label 0. SimLabelCanary: batches where all records are from $[0, 9]$ and differ in one random record. Since the baseline attack is 50%, the plot shows that FP attack is successful on DiffLabelCanary where the canary record comes from the same distribution as MNIST data but different from other records in the batch. Here, $\delta = 10^{-5}$.

- DiffLabelCanary: the batch is composed of records with the same labels in $[1, 9]$, and the canary record of its neighboring batch is a record with label 0.

Though DiffLabelCanary is handcrafted, it represents an example of a batch that should be protected by DP guarantees since all records are drawn from the same distribution.

2) Attack Results: The attack on DP-SGD follows the same procedure as the attack in Section V-B since it uses Gauss_pytorch. Note that the attack against DP-SGD naturally reveals sequential (cached) samples from an implementation of the normal distribution since the attacker observes application of noise to gradients of all d parameters of the model, i.e., to d computations that all use independent noise draws.

The adversary calculates $f(B)$ and $f(B')$ since we assume the attacker knows the records in B and B' and is trying to determine the presence of the canary. Since there are 26,010 gradients in y, we evaluate the attack on each to see if it lies in support of $f(B)$ or $f(B')$. We extract s_1 and s_2, s'_1 and s'_2 from $y - f(B)$ and $y - f(B')$ respectively for each. We then search for their neighboring values, and check whether any of them support gradients in y. We say y is in support of $f(B)$ if there are more gradients that support $f(B)$ than $f(B')$ and similarly for $f(B')$.

The experimental results are presented in Figure 3. For DiffLabelCanary the attacker's succes rates is always better than a random guess for $\epsilon < 0.68$. Similar to previous results in this section, the attack success rate is much higher than the theoretical (ϵ, δ)-DP bound on failure of $\delta = 10^{-5}$ would suggest.

We observe that the adversary cannot distinguish $f(B)$ and $f(B')$ when SimLabelCanary is used, that is, when the canary is similar to all the other records in the batch. The reason is that the gradients for B' and B are close to each other and hence, even when the noise is added the two stay relatively close to each other, hence, the range of "attackable" floating-point they land on is similar. On the other hand, for DiffLabelCanary the canary record has a very different distribution from records in B, thus the gradients are further apart which shifts them into ranges of varying number of floating points. It is important to note that the difference between the gradients (i.e., sensitivity) is protected by the DP mechanism based on a theoretical normal distribution using real values. However, this difference is large enough that once the noise is added, the noisy values are also shifted to ranges of floating points where one has more attackable values than the other. We observe a relative increase in gradient norm of 1.34 when using DiffLabelCanary compared to SimLabelCanary.

We also observe that the success rate increases when σ increases, and correspondingly ϵ decreases. This is counter-intuitive as the magnitude of noise increases as ϵ decreases. The same observation was made by Mironov for the Laplace distribution. The reason again relates to the floating-point range in which noisy gradients land. We also note that compared to attacks in V-B, the attacker has more chances to observe values in support of one dataset than the other, since it has d gradients to attack as opposed to one result.

VI. Discrete and Approximate Distribution Sampling

Discrete distributions aim to avoid privacy leakage from floating-point representation, while retaining the privacy and utility properties of their continuous counterparts. In this section we show that naïve implementation of such discrete distributions suffers from a timing side channel attack: by measuring the time the sampling algorithm takes to draw noise from such a distribution, the adversary is able to determine the magnitude of this noise and, hence, invalidate the guarantees of differential privacy.

a) Discrete Laplace and Gaussian: Canonne *et al.* [12] study the discrete Gaussian mechanism and its properties. They demonstrate that it provides the same level of privacy and utility as the continuous Gaussian. Their sampler for Laplace and Gaussian uses the geometric distribution where the corresponding samples preserve the magnitude of the noise drawn from the geometric distribution. Unfortunately, the running time of this geometric distribution sampler, if not implemented carefully, reveals the magnitude of its noise.

Canonne *et al.* [12] describe sampling from geometric distributions (Algorithm 2 in their paper) using Bernoulli samples. Recall that the geometric distribution measures the probability of taking n Bernoulli trials to obtain a first success. Their pseudo-code to simulate this process proceeds as follows. It samples the Bernoulli distribution until the first success while incrementing a counter of failures. Once the success is observed, the counter is returned as a sample n of a geometric distribution. Hence, the number of times Bernoulli sampler is invoked linearly correlates with the magnitude of the sample n. This is the source of the timing side-channel where the time is correlated with a secretly drawn value.

The Laplace distribution [12] is based on a linear transformation of the sample drawn from the geometric distribution, preserving its magnitude. In turn, the discrete Gaussian mechanism with standard deviation of σ in [12] uses the Laplace mechanism with parameter $\lfloor\sigma\rfloor + 1$. Laplace noise is returned as-is via rejection sampling with a carefully chosen probability that produces the exact discrete distribution. However, since Laplace noise is returned as-is, the Gaussian sample has the same magnitude as Laplace and hence as the geometric distribution.

In summary, the time it takes to run discrete Laplace or discrete Gaussian sampling is correlated with the magnitude of the sample they return, and hence the noise they add to their respective DP mechanisms. Though the authors state that their algorithms may suffer from timing attacks, they attribute them to rejection sampling noting that this "reveals nothing about the accepted candidate". However, as we argue above and experimentally show in the next section, the subroutine used to draw from the geometric distribution is the one that creates the timing side channel.

b) Approximate Laplace: In parallel to the work by Canonne *et al.*, the Differential Privacy Team at Google [13] proposed an algorithm for approximate Laplace in their report of the library implementation for differential privacy [3]. Their sampler makes a draw from the geometric distribution and then scales it using a resolution parameter based on ϵ. The paper does not specify how the geometric distribution is implemented. Upon examination of the code in the library [3], we observed that geometric sampling is not based on drawing Bernoulli samples as in [12]. However, its runtime still linearly depends on the value being sampled and hence also suffers from a timing side-channel. Specifically, the implementation of [13], performs a binary search, where the distribution support region is split proportional to probability mass and is guided by a sequence of uniform random values. Since the search is longer for events with smaller probability, the time to "find" larger values in the case of drawing geometric random variables takes longer. As a result this also creates a timing side channel that reveals the magnitude of the drawn noise, even though conceptually the technique for drawing from the geometric distribution is different from [12].

VII. Experiments: Timing Attacks on Discrete Distributions

We evaluate the discrete Laplace and Gaussian using the implementation in [14] that accompanied the work by Canonne *et al.* [12] and the Laplace implementation from Google [13], referred to as implementations I and II respectively (see disclosure in Section I). We conduct two sets of experiments to show that 1) both implementations are amenable to timing attacks; 2) a DP algorithm that uses these implementations, as a consequence, is also amenable to a timing attack.

A. Experimental Setup

We run the discrete samplers on a single core of an Intel Xeon Platinum 8180M, which runs a 64-bit Ubuntu Linux 16.04.1 with kernel version 4.15.0-142. There are no other running processes on this core, so the interference on timing measurement is minimized. We measure the overall time of the sampling algorithm on invocation and exit with nano second precision using `time.process_time_ns()` for Implementation I written in Python and `System.nanoTime()` for Implementation II written in Java.

B. Timing of Discrete Samplers

We first measure the time it takes to generate the noise from each implementation and average it over more than 1 million trials for each sampled value in a truncated region. Since both Gaussian and Laplace are symmetrical distributions, the time it takes to generate positive and negative noise is also symmetric. In the implementation, the sign is determined independently from the (positive) geometric noise magnitude.

In Figure 4 we plot the average time it takes to draw absolute values of the noise. We used $\sigma = 19$ for Gaussian I, $\lambda = 8$ for Laplace I, and $\lambda = \frac{8}{\ln 3}$ for Laplace II. We observe that the absolute magnitude of noise has a positive linear relationship with time to generate noise from all implementations.

Based on the above relationship between noise magnitude and generation time, we implement our attack as follows. The attacker computes average time t_i to generate absolute values of integer noise $i \in [0, 9]$. It then measures the time t_j it takes to generate an unknown noise j sample and chooses i that has the closest time to t_j as its guess:

$$j_{\mathsf{guess}} = \arg\min_{i \in [0,9]}(|t_j - t_i|) \tag{3}$$

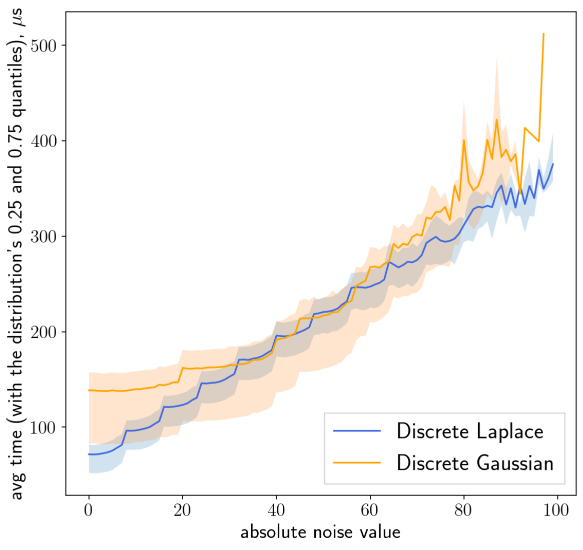
We ran 100,000 trials for each sampler to evaluate the accuracy of our timing attack. We evaluate accuracy in two standards: exact match and approximate match within ± 1 from the correct value:

- exact match: $|j| = j_{\mathsf{guess}}$
- approximate match: $-1 \leq j_{\mathsf{guess}} - |j| \leq 1$

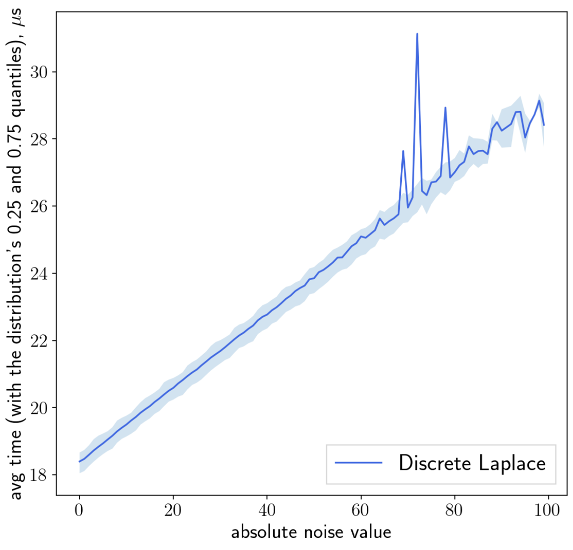
The attacker can use exact match as follows. Recall that the attacker, given a DP output y and j_{guess} where $y = f(D) + j$ is trying to guess the unprotected value of $f(D)$. Let the co-domain of f have integer support $[-X, X]$, known to the adversary as it knows f and domain of f, $\mathcal{D}$. For exact match though our attack cannot guess whether the noise is positive or negative, this still harms differential privacy. Specifically, it reduces the original guess of the attack on the noise value from $1/(2X + 1)$ to $1/2$.

For the approximate guess, the attacker knows that j could have been one of six values:

$$\pm(j_{\mathsf{guess}} - 1), \pm j_{\mathsf{guess}}, \pm(j_{\mathsf{guess}} + 1)$$



(a)



(b)

Fig. 4: Average time in μs (with the distribution's 0.25 and 0.75 quantiles) to generate absolute noise using two implementations of discrete distributions: (a) Discrete Laplace and Gaussian from Implementation I [14] (avg. over 10 million trials). (b) Discrete Laplace from Implementation II [3] (avg. over 20 million trials). The plots show a linear relationship between the absolute noise value and time it takes to generate it. Gaussian I uses $\sigma = 19$, Laplace I uses $\lambda = 8$ and Laplace II uses $\lambda = \frac{8}{\ln 3}$.

Implementation	Parameters	Match	Accuracy
Gaussian I [14]	$\sigma = 2$	exact	24.4%
		approximate	56.2%
	$\sigma = 4$	exact	13.6%
		approximate	39.9%
Laplace I [14]	$\lambda = 1$	exact	42.1%
		approximate	84.2%
	$\lambda = 3$	exact	24.7%
		approximate	61.5%
Laplace II [3]	$\lambda = \frac{1}{\ln 3}$	exact	42.0%
		approximate	89.0%
	$\lambda = \frac{3}{\ln 3}$	exact	17.0%
		approximate	44.0%
	$\lambda = \frac{1}{\ln 2}$	exact	32.0%
		approximate	71.0%

TABLE I: Success rates of timing side channel attacks against implementations of discrete distributions. The baseline accuracy based on a random guess for exact and approximate match is 10% and 33%, respectively. We observe that the attacks are always well above the baseline attack accuracy.

This allows it to determine that $f(D)$ must be either $y \pm (j_{\text{guess}} - 1)$, $y \pm j_{\text{guess}}$ or $y \pm (j_{\text{guess}} + 1)$. Hence, its guess is reduced from $1/(2X+1)$ to $1/6$. As an example, suppose the attacker is trying to distinguish between $f(D) = 30$ and $f(D') = 60$. If it observes, $y = 20$ and $j_{\text{guess}} = 9$. It knows that f must have been computed on D and not D'.

The results are summarized in Table I where we measure the accuracy for samples in the range $[-9, 9]$ for several parameters where ϵ ranges between 0.5 and 2 and $\delta = 10^{-5}$. Since we evaluate the absolute values of the sample, the baseline accuracy based on a random guess for exact and approximate match is 10% and 33%, respectively. We observe that the attacks are always well above the baseline accuracy. This indicates that if an adversary can observe how long the sampling algorithm takes to generate noise used in a DP mechanism, then it can guess the relative magnitude of this noise for samplers based on geometric noise generation.

For Implementation I, the discrete Laplace mechanism is more vulnerable to timing attacks than the discrete Gaussian. This is likely due to rejection sampling that Gaussian implementation adds to the process. Rejection sampling adds stochasticity to which Laplace sample, among several that are drawn, is returned. For example, the attacker cannot distinguish between the following two executions:

1) In the first execution, a large Laplace noise value is generated but then rejected, while a small Laplace noise value generated next is returned as a result.
2) In the second execution, the opposite happens where a small Laplace value is rejected first and then a larger Laplace value is generated and returned as a result.

Execution times will be similar for both cases. Indeed, the authors of [13] also proposes a discrete Gaussian based on Binomial sampling and rejection sampling and their implementation [3] is not amenable to our attack.

The success rate of our attack decreases with increasing σ and λ. We note that with higher parameters, larger noise values are more likely, while the attack is more successful for smaller values, hence, overall accuracy decreases. For example, with $\lambda = 3$ for Laplace I, the accuracy for $i \in [0, 4]$ is 27.1%, and accuracy for $i \in [5, 9]$ is 12.5%.

For Laplace II we observe a similar trend as for Laplace I even though the geometric distribution is sampled using a different procedure described in the previous section.

C. Timing Attack on Private Sum

Based on the results of the previous section, we conduct a timing attack on real data using the German Credit Dataset [33] used in Section V-B1. We model the setting where the private sum of the credit attribute of the dataset is computed (e.g., an analyst queries a DP-protected database as in the DB setting in Section III). The attacker is trying to guess the non-private sum using query's response and the time it takes for the query to return. We put a limit that each individual can have at most 5000 credits (which determines the sensitivity of the query). We use private sum computation which mirrors private count in Section V-B1 except f is a sum and s is sampled from Laplace II [3]. The neighboring datasets D and D' that our attack tries to distinguish differ on a single record whose credit is 5000 in D and 0 in D'. We note that the query is different from that in Section V-B1 since the timing attack is not as effective for queries with small sensitivity, while the FP attack is effective against queries that return values close to 0.

To conduct the attack, we first collect timing data of private sum for different noise magnitudes. Similar to experiments in the previous subsection, we observe a linear relationship between noise magnitude and time cost albeit now this time includes computing the sum and sampling (Figure 8 in Appendix).

Our attack proceeds by measuring the time t of a DP algorithm to complete. The attacker then uses t and the output y it receives to determine if it was D or D' used in the computation. Note that the noise magnitude should be $s = |y - \sum_D x|$ for D and $s' = |y - \sum_{D'} x|$ for D'.

The attacker makes a guess on the magnitude of the noise using the timing data it has collected above and Equation 3. Let s_g be its guess. It then compares s_g with s and s' and chooses the closest one as its guess. That is, it chooses D if $|s - s_g| < |s' - s_g|$ and D' otherwise.

We plot the attack results in Figure 5 for ϵ in the range $[1, 10]$. We observe that attack success rate increases with higher ϵ, with success rate of 69.15% for $\epsilon = 1$. Our intuition for the above trend is due to the range of noise in which the attacker needs to make its guess. That is, for smaller noise scale (i.e., high ϵ), the output noise range is small as well and hence attacker has less number of noises to assign observed time to. For example, with $\Delta = 5000$, noise magnitudes are mainly distributed in $[0, 15000]$ when $\epsilon = 1$, while noise magnitudes are mainly distributed in $[0, 2500]$ when $\epsilon = 10$.

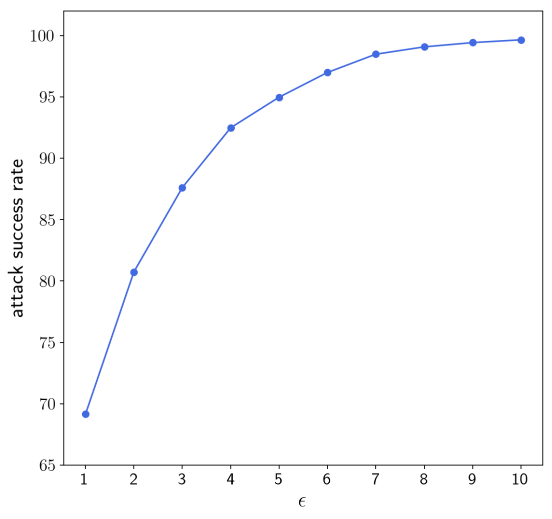


Fig. 5: Attack on private sum of the credit attribute of the German Credit Dataset [33], with Laplace II [3]. The attack success rate under different privacy budget $\epsilon \in [1, 10]$ and sensitivity $\Delta = 5000$ (measured over 1 million trials for each ϵ). Success means the private sum created from D is successfully concluded to be in support of D and not D'.

The timing attack has two limitations. First, it assumes that the time (and its variance) to compute f is not much larger than that of sampling, as otherwise the microseconds difference may not be observable. Second, the attack works for one-dimensional functions f as the attacker can measure the time of a single noise sample. Hence, the attack will not be as successful if the attacker were to observe the time it takes to draw multiple samples, as is the case for DP-SGD.

VIII. Mitigation Strategies

We discuss mitigation strategies for both of our attacks.

A. Defenses Against Floating-Point Attacks

Mironov [6] proposed the snapping mechanism to alleviate the FP attack by carefully truncating and rounding an output of a DP mechanism that was implemented using floating points. However, the privacy and utility of the overall mechanism decreases [13], [12].

Our attack against the Box-Muller and polar methods assumes that the attacker observes two of their samples (recall that the Ziggurat method generates only one sample). That is, the attacker gets access to the second (cached) value (e.g., when a query returns an answer to a d-dimensional query such as a histogram or ML model parameters). Without the second value in Equation (2) the attacker has to resort to checking all possible values in a brute-force manner. A potential mitigation is therefore to generate new samples on each call and disregard the second value.

We also observe that the implementation of DP-SGD adds noise to the batch directly. Instead it could potentially add several samples of noise and then average the result, since

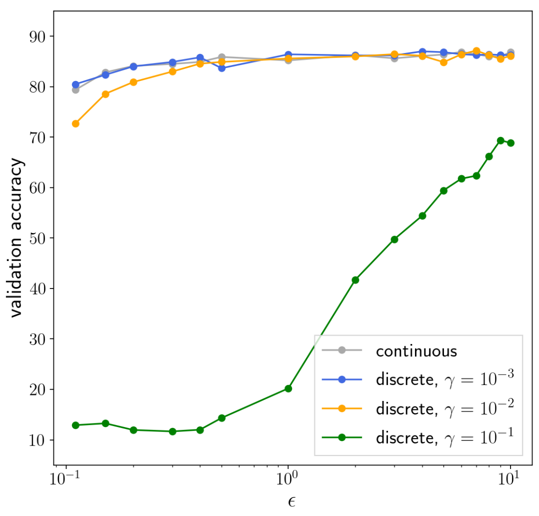


Fig. 6: Accuracy of the ML model on MNIST dataset, trained with DP-SGD with continuous and discrete Gaussian samplers with discretization parameters $\gamma \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ for $\epsilon \in (0.1, 10]$.

an average of Gaussian noise is still Gaussian. This would make it harder for an adversary to extract sample-level values needed for the FP attack as described in Section IV. However, this heuristic may still be susceptible to attacks as it also uses floating-point representation.

Discrete distributions [12], [13], [39] have been proposed as a mitigation against floating-point attacks since they avoid floating points or bound their effect on privacy in (ϵ, δ) parameters. However, as we showed in the previous section, they may suffer from other side-channel attacks and thus should also be carefully implemented. Nevertheless in this section we evaluate them as a mitigation for attacks in Section V-C and measure their effect on accuracy of DP-SGD.

We implement DP-SGD with discrete Gaussian by discretizing the gradient $g \in \mathbb{R}^d$ as described in [40] who use discrete Gaussian for training models in a Federated Learning setting. Appendix C provides further details.

We use the same model and MNIST dataset as in Section V-C to evaluate the performance of models trained with discrete Gaussian for a discretization parameter $\gamma \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ and privacy budget $\epsilon \in (0.1, 10]$. We use the implementation of discrete Gaussian I [14].

We compare accuracy of the models with discrete and continuous implementations in Figure 6. We observe that the performance of models trained with discrete Gaussian matches the performance of models trained with continuous Gaussian, given small enough γ, such as 10^{-3}. Smaller γ allows the discretization to be done on a finer grid $\gamma\mathbb{Z}$, thus gradient calculation is not affected by rounding.

B. Defenses Against Timing Attacks

The most effective mitigation against timing attacks is to ensure constant time execution. Application-independent approaches to do so include compiler-based code transformations [41], [42] while other generic defense strategies are based on padding either by padding execution time to a constant time or adding a random delay [43].

In our setting, one could choose a sufficiently large time threshold and run noise generation mechanism until then, independent of the noise being drawn and the time that takes to produce it. If an attacker is stronger than that considered in this paper and can perform microarchitectural observations, then execution of any padded code has to be made secret-independent. For example, if the attacker can measure the time of memory accesses, the counter of the number of trials needs to be accessed regardless of whether heads or tails was drawn in a Bernoulli trial. The downside of padding is efficiency as all execution would take maximum time. The failure to draw noise within the threshold time can then be accounted for in δ, the failure probability of DP [12]. An alternative is to use a truncated version of the geometric distribution [7] in order to avoid values that are impossible to sample on a finite computer.

We evaluate padding as a mitigation against our attack on private sum in Section VII-C. Padding can be implemented via two approaches. First, after a successful trial is encountered, we record its trial number and continue drawing "blank" Bernoulli samples, till the maximum noise threshold is reached. In the second approach, once a successful trial is encountered, a time delay can be added to reach some maximum time threshold. We choose the maximum threshold to be 41μ since we observed that 99.5% of noise magnitudes are distributed in $[0, 2677]$, with average and maximum execution time of 37.1μ and 40.3μ, respectively, when $\epsilon = 10$ and $\Delta = 5000$. We note that these estimates are not dependent on the data but only on the sampler and the parameters of the noise. Padding until a time threshold increases the total execution by 10.5%. For medium number of queries this is an acceptable overhead given the small magnitude of the overall time (in μs). Note that these estimates will be different for other implementations.

As another mitigation strategy we propose a technique based on batching and caching. The method generates k random values offline and saves them. It returns one value for each call to the distribution function. It proceeds this way until all cached values are used and then restarts the process by generating the next k samples. Samples could be generated online and shuffled to disconnect noise from their timing, as suggested in [44]. However, the attacker may still measure the range of possibles times.

In summary, discrete distribution sampling implemented in constant time or where the timing of sampling is not observable (i.e., generated offline) appears to be the best approach for defending against attacks discussed in this paper.

IX. Related Work

The implementation of differential privacy via Laplace mechanism has been demonstrated to be flawed, due to finite-precision representations of floating-point values [6]. In this paper, we demonstrate that implementations of the Gaussian mechanism suffer from the same attack, with adjustments to the attack process. In [45] the authors demonstrate an attack against DP mechanisms that use finite-precision representations, and propose a mitigation strategy. Recently Ilvento [46] has explored practical considerations and pitfalls of implementing the exponential mechanism using floating-point arithmetic. They show that such implementations are also susceptible to attacks and propose a solution using a base-2 version of the exponential mechanism.

Timing attacks against DP mechanisms have been explored in [7] and [8]. However, they differ from the attacks described in this paper as they do not exploit the timing discrepancies introduced by distribution sampling. In [7], the authors observe that the mechanism implementation may suffer from timing attacks (e.g., because it performs conditional execution based on a secret). As a mitigation authors propose constant time execution for the mechanism, without consideration of noise generation. On the other hand, Andrysco *et al.* [8] exploit the difference in timing of floating-point arithmetic operations (e.g., multiplication by zero takes observably less time than multiplication by a non-zero value). They also show that DP mechanisms (including [7]) are susceptible to information leakage by using floating-point instructions whose running time depends on their operands.

Balcer and Vadhan [39] outline shortcomings of implementing differentially-private mechanisms on finite-precision computers, including a discussion of floating-point representations and sampling from distributions with infinite support. The authors propose a polynomial-time discrete method for answering approximate histograms. Their method is based on a bounded (or truncated) geometric distribution. Though a full implementation is not provided, the authors suggest that the distribution can be sampled via inverse transform sampling using binary search over the support range using cumulative distribution function F to guide the search. That is, given a uniform random value p sampled uniformly from $(0, 1]$, find smallest value x from the support of the geometric distribution such that $F(x) \geq p$. If performed naïvely, such an approach could reveal the magnitude of the noise of the geometric distribution since the search will take longer for "less likely" values (i.e., larger p) and would therefore suffer from the same timing channel as described in Section VI.

In independent and parallel work [47], the authors describe a theoretical attack against the Box-Muller method of sampling from the Gaussian distribution in a similar manner to our FP attack. However, they do not provide experiments validating the attack's efficacy against existing implementations. The authors propose a mitigation strategy similar to the one mentioned in Section VIII based on computing a Gaussian sample from multiple samples, and analyze its robustness. Their work does not consider timing attacks.

Stepping away from differential privacy, Gaussian samplers have been also widely used in schemes for digital signatures, public key encryption, and key exchange based on lattice based cryptography [48], [49], [50]. Such cryptographic primitives often rely on a multi-dimensional discrete Gaussian which is approximated by a distribution that is statistically close to the desired distribution. Several sampling mechanisms have been shown to suffer from cache-based side-channel attacks [51] (i.e., based on memory accesses of the underlying algorithm) and power analysis [44]. Defenses based on constant-time execution [52], [53], [54] and shuffling [44] have been proposed to protect against some of these attacks including timing. Though some techniques proposed for hardening the code in this space can be used for protecting samplers for DP (Section VIII), direct use of such samplers for DP is not straightforward. This follows from the observation that discrete variants in this area are (only) statistically close to the desired discrete distribution. As a result, composition-based analysis developed for analyzing cumulative loss of multiple mechanisms based on Gaussian noise [17], [35] cannot be used directly.

X. Conclusion

In this paper we highlight two implementation flaws of differentially private (DP) algorithms. We first show that the widely used Gaussian mechanism suffers from a floating-point (FP) attack against implementations of normal distribution sampling, similar to vulnerabilities of the Laplace mechanisms as demonstrated in 2011 by Mironov. We empirically demonstrate that implementations in `NumPy`, `PyTorch` and `Go`, including those used in implementation of open-source DP libraries are susceptible to the attack, hence violating their privacy guarantees. Though some researchers have speculated that the Gaussian mechanism may be susceptible to FP attacks, this is the first work to provide a comprehensive evaluation showing that it is feasible in practice.

In the second part of the paper we show that implementations of discrete Laplace and Gaussian mechanisms — proposed as a remedy to the FP attack against their continuous counterparts — are themselves vulnerable to another side-channel due to timing. That is, we show that implementations of such discrete variants, including a DP library by Google, exhibit the time that is correlated with the magnitude of the secret random noise. Our work re-iterates the importance of careful implementation of DP mechanisms in order to maintain their theoretical guarantees in practice.

Acknowledgment

The authors are grateful to the anonymous reviewers for their feedback that helped improve the paper. This work was supported in part by a Facebook Research Grant and the joint CATCH MURI-AUSMURI. The first author is supported by the University of Melbourne research scholarship (MRS) scheme. We thank Thomas Steinke for pointing us to the Ziggurat method in Go and Clément Canonne for insightful discussions on statistically-close Gaussian samplers.

References

[1] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in ACM Symposium on Operating Systems Principles (SOSP), 2017.

[2] G. Andrew, S. Chien, and N. Papernot, “TensorFlow Privacy,” https://github.com/tensorflow/privacy, 2019, [Online; accessed 01-Dec-2021].
[3] “Google Differential Privacy,” https://github.com/google/differential-privacy, 2021, [Online; accessed 01-Dec-2021].
[4] “Opacus,” https://github.com/pytorch/opacus, 2020, [Online; accessed 01-Dec-2021].
[5] “Diffprivlib,” https://github.com/IBM/differential-privacy-library, 2021, [Online; accessed 01-Dec-2021].
[6] I. Mironov, “On significance of the least significant bits for differential privacy,” in ACM Conference on Computer and Communications Security (CCS). ACM, 2012, pp. 650–661.
[7] A. Haeberlen, B. C. Pierce, and A. Narayan, “Differential privacy under fire,” in USENIX Security Symposium, 2011, p. 33.
[8] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham, “On subnormal floating point and abnormal timing,” in IEEE Symposium on Security and Privacy (S&P). IEEE Computer Society, 2015, pp. 623–639. [Online]. Available: https://doi.org/10.1109/SP.2015.44
[9] 30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017. IEEE Computer Society, 2017. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8048777
[10] M. Bun, C. Dwork, G. N. Rothblum, and T. Steinke, “Composable and versatile privacy via truncated cdp,” in Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, 2018, pp. 74–86.
[11] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in ACM Conference on Computer and Communications Security (CCS). ACM, 2016, pp. 308–318.
[12] C. L. Canonne, G. Kamath, and T. Steinke, “The discrete Gaussian for differential privacy.” NeurIPS, 2020.
[13] Google, “Secure noise generation,” https://github.com/google/differential-privacy/blob/master/common_docs/Secure_Noise_Generation.pdf, 2020.
[14] “The Discrete Gaussian for Differential Privacy,” https://github.com/IBM/discrete-gaussian-differential-privacy, 2020, [Online; accessed 01-Dec-2021].
[15] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in Theory of Cryptography Conference (TCC), 2006, pp. 265–284.
[16] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” Foundations and Trends in Theoretical Computer Science, vol. 9, no. 3-4, pp. 211–407, 2014. [Online]. Available: https://doi.org/10.1561/0400000042
[17] C. Dwork and G. N. Rothblum, “Concentrated differential privacy,” CoRR, vol. abs/1603.01887, 2016. [Online]. Available: http://arxiv.org/abs/1603.01887
[18] I. 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. [Online]. Available: https://doi.org/10.1109/CSF.2017.11
[19] R. Bassily, A. D. Smith, and A. Thakurta, “Private empirical risk minimization: Efficient algorithms and tight error bounds,” in Symposium on Foundations of Computer Science (FOCS), 2014.
[20] S. Song, K. Chaudhuri, and A. D. Sarwate, “Stochastic gradient descent with differentially private updates,” in 2013 IEEE Global Conference on Signal and Information Processing, Dec 2013, pp. 245–248.
[21] Y.-X. Wang, S. Fienberg, and A. Smola, “Privacy for free: Posterior sampling and stochastic gradient Monte Carlo,” in International Conference on Machine Learning (ICML), 2015.
[22] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” in International Conference on Learning Representations (ICLR), 2018.
[23] L. Yu, L. Liu, C. Pu, M. Gursoy, and S. Truex, “Differentially private model publishing for deep learning,” in IEEE Symposium on Security and Privacy (S&P), 2019.
[24] J. M. Abowd, “The u.s. census bureau adopts differential privacy,” in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD ’18, 2018.
[25] N. Johnson, J. P. Near, and D. Song, “Towards practical differential privacy for SQL queries,” PVLDB, vol. 11, no. 5, pp. 526–539, Jan. 2018.
[26] F. D. McSherry, “Privacy integrated queries: An extensible platform for privacy-preserving data analysis,” in SIGMOD, 2009.
[27] J. Allen, B. Ding, J. Kulkarni, H. Nori, O. Ohrimenko, and S. Yekhanin, “An algorithmic framework for differentially private data analysis on trusted processors,” in Conference on Neural Information Processing Systems (NeurIPS), 2019.
[28] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds,” in ACM Conference on Computer and Communications Security (CCS), 2009.
[29] T. Van Goethem, C. Pöpper, W. Joosen, and M. Vanhoef, “Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections,” in USENIX Security Symposium, 2020.
[30] G. Marsaglia and T. A. Bray, “A convenient method for generating normal variables,” SIAM Review, vol. 6, no. 3, p. 260–264, 1964.
[31] G. Marsaglia and W. W. Tsang, “The ziggurat method for generating random variables,” Journal of Statistical Software, vol. 5, no. 8, 2000.
[32] G. Marsaglia, “Generating a variable from the tail of the normal distribution,” Technometrics, vol. 6, no. 1, pp. 101–102, 1964. [Online]. Available: https://doi.org/10.1080/00401706.1964.10490150
[33] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[34] B. Balle and Y.-X. Wang, “Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising,” in ACM Conference on Computer and Communications Security (CCS). ICML, 2018, pp. 1–23.
[35] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” Found. Trends Theor. Comput. Sci., vol. 9, Aug. 2014.
[36] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce, and A. Roth, “Differential privacy: An economic method for choosing epsilon,” in 2014 IEEE 27th Computer Security Foundations Symposium. IEEE, 2014, pp. 398–410.
[37] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[38] Z. Bu, J. Dong, Q. Long, and W. J. Su, “Deep learning with Gaussian differential privacy,” CoRR, vol. abs/1911.11607, 2019. [Online]. Available: http://arxiv.org/abs/1911.11607
[39] V. Balcer and S. P. Vadhan, “Differential privacy on finite computers,” in 9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA, ser. LIPIcs, A. R. Karlin, Ed., vol. 94. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 43:1–43:21. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2018.43
[40] P. Kairouz, Z. Liu, and T. Steinke, “The distributed discrete gaussian mechanism for federated learning with secure aggregation,” in Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 5201–5212. [Online]. Available: http://proceedings.mlr.press/v139/kairouz21a.html
[41] J. V. Cleemput, B. Coppens, and B. De Sutter, “Compiler mitigations for time attacks on modern x86 processors,” ACM Trans. Archit. Code Optim., vol. 8, no. 4, jan 2012. [Online]. Available: https://doi.org/10.1145/2086696.2086702
[42] M. Wu, S. Guo, P. Schaumont, and C. Wang, “Eliminating timing side-channel leaks using program repair,” in Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 15–26. [Online]. Available: https://doi.org/10.1145/3213846.3213851
[43] A. Askarov, D. Zhang, and A. C. Myers, “Predictive black-box mitigation of timing channels,” in ACM Conference on Computer and Communications Security (CCS), 2010, pp. 297–307.
[44] S. Sharma, A. Bag, and D. Mukhopadhyay, “Compact and secure generic discrete gaussian sampler based on hw/sw co-design,” in 2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2020, pp. 1–6.
[45] I. Gazeau, D. Miller, and C. Palamidessi, “Preserving differential privacy under finite-precision semantics,” in Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2013, Rome, Italy, March 23-24, 2013, ser. EPTCS, L. Bortolussi and H. Wiklicky, Eds., vol. 117, 2013, pp. 1–18. [Online]. Available: https://doi.org/10.4204/EPTCS.117.1
[46] C. Ilvento, “Implementing the exponential mechanism with base-2 differential privacy,” in ACM Conference on Computer and Communications Security (CCS), ser. CCS ’20. New York, NY, USA:

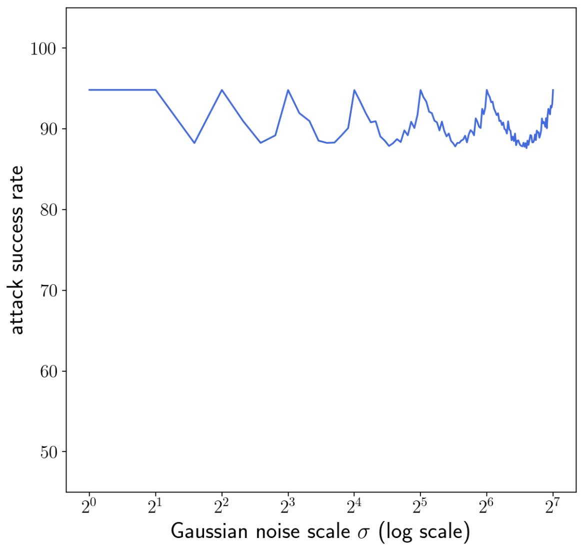


Fig. 7: FP attack success rate on private count where the count is protected with the Gauss_go Gaussian sampler across $\sigma \in [1, 128]$ with fixed $\delta = 10^{-5}$ and function sensitivity $\Delta = 1$. Baseline (random) attack success is 50%. Success rate peaks at $\sigma \in \{2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7\}$.

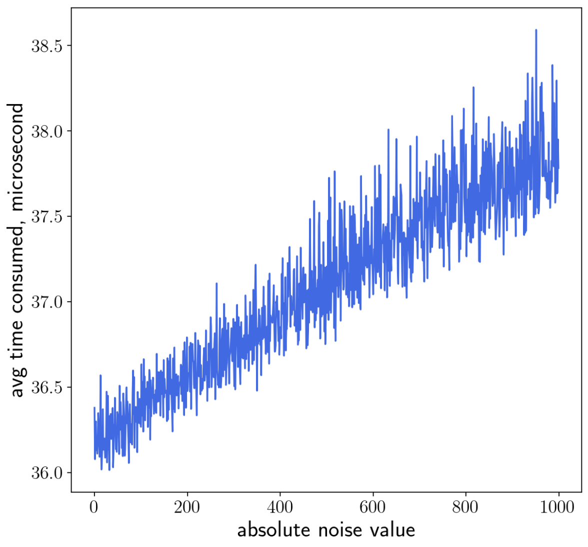


Fig. 8: Execution time of private sum over the credit attribute of the German Credit Dataset [33], with Laplace II [3], over a range of noise magnitudes, with $\epsilon = 10$ and sensitivity $\Delta = 5000$ (avg. over 50 million trials).

Association for Computing Machinery, 2020, pp. 717–742. [Online]. Available: https://doi.org/10.1145/3372297.3417269

[47] N. Holohan and S. Braghin, "Secure random sampling in differential privacy," CoRR, vol. abs/2107.10138, 2021. [Online]. Available: https://arxiv.org/abs/2107.10138

[48] J. Folláth, "Gaussian sampling in lattice based cryptography," Tatra Mountains Mathematical Publications, vol. 60, no. 1, pp. 1–23, 2015. [Online]. Available: https://doi.org/10.2478/tmmp-2014-0022

[49] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," J. ACM, vol. 60, no. 6, nov 2013. [Online]. Available: https://doi.org/10.1145/2535925

[50] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in Advances in Cryptology – CRYPTO 2013, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 40–56.

[51] L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom, "Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme," in Cryptographic Hardware and Embedded Systems – CHES 2016, B. Gierlichs and A. Y. Poschmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 323–345.

[52] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill, "On practical discrete gaussian samplers for lattice-based cryptography," IEEE Transactions on Computers, vol. 67, pp. 322–334, 2018.

[53] D. Micciancio and M. Walter, "Gaussian sampling over the integers: Efficient, generic, constant-time," Cryptology ePrint Archive, Report 2017/259, 2017, https://ia.cr/2017/259.

[54] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Constant-time discrete gaussian sampling," IEEE Transactions on Computers, vol. 67, no. 11, pp. 1561–1571, 2018.

[55] D. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, "A hardware Gaussian noise generator using the Box-Muller method and its error analysis," IEEE Trans. Computers, vol. 55, no. 6, pp. 659–671, 2006. [Online]. Available: https://doi.org/10.1109/TC.2006.81

Appendix

A. *Box-Muller Method*

The Box-Muller method [55] is a computational method that generates samples of the standard normal distribution from uniformly distributed random values. It operates as follows:

1) Choose independent uniform random values x_1 and x_2 from $(0, 1)$ using RandomFP.
2) Set $r^2 \leftarrow -2 \ln x_1$ and $\Theta \leftarrow 2\pi x_2$.
3) Return $s_1 \leftarrow r \cos(\Theta)$.

This procedure can be used to generate two independent normal samples: s_1, as above, and $s_2 \leftarrow r \sin(\Theta)$.

B. *DP Gradient Computation*

We now recall how f and Δ_f are determined for the DP-SGD mechanism.

$$f(B) = \frac{1}{S} \sum_{i=1}^{S} g(b_i) \tag{4}$$

where $B = [b_1, b_2, \ldots, b_S]$ and b_i is a record in B, g calculates per-record gradient with per-record clipping with respect to clipping norm L. Note that since the model has $d = 26,010$ parameters and each batch is used to update them all, $f(B) \in \mathbb{R}^d$ contains a gradient for each parameter of the model.

a) Sensitivity Analysis: The sensitivity of DP-SGD is the maximum L_2 distance between any pair of $f(B)$ and $f(B')$. Clipping norm L dictates the largest L_2 norm of any record gradient. Thus, for any $f(B)$ and $f(B')$, we can have:

$$\begin{aligned}\|f(B') - f(B)\|_2 &\leq \frac{1}{S}\|g(b_c) - g(b_r)\|_2 \\ &\leq \frac{1}{S}(\|g(b_c)\|_2 + \|g(b_r)\|_2) \leq \frac{2L}{S}\end{aligned}$$

TABLE II: The range of attack rate and accuracy across $\epsilon \in [1, 20]$. Attack rate refers to the rate at which the attacker can make a gues. Attack accuracy measures how many of these guesses are correct. The two implementations of polar method and Box-Muller, Gauss_numpy and Gauss_pytorch, respectively, have different attack rates, though the accuracy of those guesses is always above 89%. The attack rate and accuracy for Gauss_go always stay above 76% and 99.9%, respectively.

Sensitivity	Gauss_numpy		Gauss_pytorch		Gauss_go	
	attack rate	attack accuracy	attack rate	attack accuracy	attack rate	attack accuracy
1	$[1.7\%, 78.2\%]$	$[92.4\%, 99.9\%]$	$[4.3\%, 92.8\%]$	$[97.7\%, 99.9\%]$	[76.4%, 89.6%]	[99.9%, 100%)
10	$[1.9\%, 76.5\%]$	$[89.6\%, 99.9\%]$	$[10.9\%, 91.9\%]$	$[99.5\%, 99.9\%]$	[76.3%, 88.8%]	[99.9%, 100%)

where b_c refers to the canary record and b_r refers to the record replaced by b_c. Since $L = 1$ and $S = 64$, the sensitivity of f is $1/32$.

DP-SGD then proceeds by computing

$$y = f(B) + \frac{1}{S} Z$$

where $Z = [Z_1, \ldots, Z_d] \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2 L^2)$, $y \in \mathbb{R}^d$, and Gauss_pytorch is used to draw noise from $\mathcal{N}$. Note that even if some record is radically different from others in the batch (e.g., like the canary used in DiffLabelCanary) its gradient is clipped at L.

C. DP-SGD Discretization

In Section VIII-A, we implement DP-SGD with discrete Gaussian by discretizing the gradient $g \in \mathbb{R}^d$ as described in [40] who use discrete Gaussian for training models in Federated Learning setting. The method proceeds as follows.

1) Clip g w.r.t. clipping norm L, $g' = g \times \min\{1, L/\|g\|_2\}$.
2) Scale g' with discretization parameter $1/\gamma$, so the discretization can be done on a finer grid $\gamma\mathbb{Z}^d$. Then randomly round each coordinate to the nearest integer, $z = \mathsf{Round}_\gamma(g')$ so that $z \in \mathbb{Z}^d$.
3) Let $G \in \mathbb{Z}^d$ consist of d independent samples from the discrete Gaussian $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2)$.
4) Add discrete noise to z and undo discretization, $z' = (z + G) \times \gamma$.

where Round_γ is the conditional randomized rounding function with discretization parameter γ. We refer the reader to [40] for details about Round_γ.

D. Ablation study

Here we investigate the rate at which the attacker can make a guess (attack rate) and how many of these guesses are correct (attack accuracy). To this end, we simulate settings where $q = 0$ and $q' = c$ and test two values of c: 1 and 10, meaning the sensitivity Δ of the underlying function is 1 and 10, respectively. We take a range of values of ϵ from 1 to 20 while keeping a fixed $\delta = 10^{-5}$. Table II shows that implementations Gauss_numpy and Gauss_pytorch have different attack rates; however, their attack accuracy is always at least 89%. For Gauss_go, we observe that the attack rate is not influenced significantly by ϵ (at least 76%), and the attack accuracy is always above 99.99%.