Precision Attacks on Differential Privacy

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Jack Mingjie Liu

Spring, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Tianhao Wang, Department of Computer Science

Precision Attacks on Differential Privacy

Jack Liu
University of Virginia
jml7ctd@virginia.edu

ABSTRACT

Differential privacy is a method used to solve the issue of collecting sensitive data from users. However, without careful considerations, the finite precision of computers can cause problems when implementing the mathematical theory into practice. Naïve implementations in fact leak full information through analysis of the least significant bits of the data. This project explores one such attack on an implementation of differential privacy in Python and NumPy. The attack was shown to have a greater than 60% success rate in uncovering the true data. This completely breaks any differential privacy guarantees for similar such implementations and serves as a reinforcement of prior work.

1 INTRODUCTION

In our digitally connected world, the demand for information as a resource for innovation can seem insatiable. Modern machine learning algorithms, for example, depend on having a wealth of data to train and iterate on. But there are a myriad of security and privacy concerns with how this data is collected, stored, and used. There have been regulations enacted to protect individuals by restricting data collectors like the European Union's General Data Protection Regulation [1]. In this landscape, the question is then how can we find a way to responsibly collect data in order to protect privacy?

One potential solution that has been proposed is differential privacy. It describes the idea of being able to determine aggregate statistics on a population without revealing too much about any individual and thus protecting their privacy. In general, this is achieved by first calculating the true statistic and then adding a noise value in order to mask the result. One advantage of differential privacy is that it is preserved through later compositions. This means it can be used to train machine learning models securely by only needing to protect the inputs with differential privacy [9]. However, differential privacy is not restricted only to theory. In fact, industry giants including Google and Facebook have utilized this principle in order to gather usage statics of their products privately[6].

However, differential privacy is not foolproof and there can be nuances involved in implementing the theory in practice. In a seminal paper on the topic, Mironov showed the existence of a floating point precision-based attack on naïve implementations of differential privacy [8]. The idea behind the attack is that computers have finite precision and are thus unable to represent the floating-point-valued noise exactly. Because of this, one is able to deduce what noise values are and are not possible. If this can be achieved, then an attacker can filter the noise out entirely leaving the unprotected true value visible. Since then, similar attacks have been studied including a paper by Jin et al. where they break a separate differential privacy algorithm as well as discuss other side channels such as timing attacks [7]. Luckily, there are also mitigation strategies discussed in both papers which can defend against the proposed attacks.

In this project, I take the idea from these previous studies and replicate their results within the Python programming language along with the popular mathematics library NumPy. I will show that without taking appropriate precautions, the finite precision limitations of the language can leak the underlying secret true values. While these attacks can be avoided, there are still many vulnerable implementations that exist on the internet [2].

2 BACKGROUND

2.1 Differential Privacy

In a groundbreaking work, Dwork et al.[5] proposed a mathematical definition for differential privacy as a property of an algorithm. It can be stated as follows:

Definition. An algorithm $\mathcal{A}\colon \mathcal{D} \to \mathcal{R}$ satisfies *ϵ-differential privacy* if for all subsets $S \subset \mathcal{R}$ and for all pairs of *adjacent* inputs $D, D' \in \mathcal{D}$:

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(D') \in S]$$

What it means for inputs to be adjacent is highly domain dependent and the granularity can depend on the level of privacy guarantee desired. This gives rise to the definition of sensitivity denoted by Δ as what amount can the underlying true statistic can differ between adjacent inputs. For example, if our statistic is the total count of records in our dataset, then the sensitivity is 1. However, if instead it was an average of salaries then the sensitivity can be much larger. In this project, we consider adjacent inputs as databases that differ in one record with sensitivity 1. An important part of this definition of differential privacy is that it necessitates that all outputs under $\mathcal{A}(D)$ also be possible under $\mathcal{A}(D')$. In our attacks on differential privacy, we will show that this does not hold for some naïve implementations. In this definition, ϵ is sometimes called the privacy budget and it can be seen as a tuning parameter that controls the level of privacy guarantee. For ϵ close to zero, the two probabilities are more tightly bounded and so there is more privacy. However, this usually results in noisier data to the point where it is no longer useful. We will see that precision attacks work at all ϵ values, but it can affect the success rate.

2.2 Laplacian Mechanism

There are many ways of adding differential privacy to an algorithm, but one of the most common and basic is the Laplacian mechanism. First, let us define the Laplacian distribution:

Definition. The (zero-mean) *Laplacian distribution* $\text{Lap}(\lambda)$ is continuous probability distribution defined by its density function

$$P_{\lambda}(x) = \frac{1}{2\lambda} \exp\left(-\frac{|x|}{\lambda}\right)$$

Another way of describing this distribution is that it is the standard exponential distribution with a sign chosen randomly. This gives us an easy way to sample from this distribution. It is a well established fact in theory that given a function f with sensitivity Δ, then the following transformation is ϵ-differentially private:

$$\mathcal{A}(D) \coloneqq f(D) + Y, \text{ where } Y \leftarrow \text{Lap}(\Delta/\epsilon)$$

While there are other mechanisms that exist with other noise distributions, the Laplacian mechanism is the focus of this project.

2.3 Random Sampling

In order to sample arbitrary random distributions on a computer, it is common to first sample from a uniform distribution to use with the appropriate inverse cumulative distribution function. Due to the limited precision of computers, sampling from the uniform distribution involves choosing a random value within $[0, 1)$ spaced by some small quanta. In the case of Python and NumPy, all values are a multiple of 2^{-53}[3]. In order to transform this into the Laplacian distribution, we apply the quantile, or inverse cumulative distribution, function which can be defined by the following:

$$sign(r) \cdot \lambda \ln(1 - 2|r|)$$

Here, r is a value drawn from the uniform random distribution shifted down by one half[4]. Due to the non-linearity of the ln function, the output is no longer evenly distributed by some fixed quanta like r was. Instead, it is possible that two values of r map to the same output, or that there is a gap of values that no r could've been the input to some output. Figure 1 below shows a graphical representation of this phenomenon.

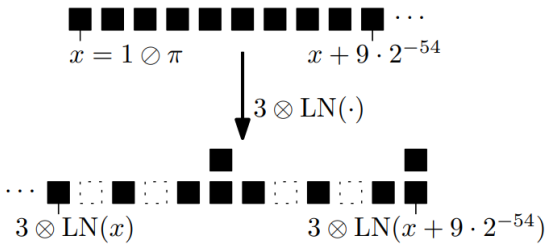

Figure 1: Gaps and overlaps when sampling from Lap(1/3)[8]

3 METHODS

The first step of this project was to identify how the library NumPy generates Laplacian noise. This was done by looking at the source code which is available publicly on github [4]. Specifically, the uniform random number generator was examined to determine the appropriate quanta size in NumPy, and the Laplacian random noise function was examined to determine the exact sampling method. These principles are crucial to the attack developed by Mironov and understanding how it is done in NumPy is critical to replicating the attack.

Next, a simple experiment was set up in order to take the output of the Laplacian mechanism and differentiate between two possible values of the underlying true data. Specifically, the goal is to determine whether some output is either $100 + \text{Lap}(\Delta/\epsilon)$ or $101 + \text{Lap}(\Delta/\epsilon)$. Here the Laplacian random values were drawn through the NumPy function `np.random.laplace`. Achieving this goal would break all differential privacy guarantees. This is because by definition all outputs must be possible in both cases for differential privacy to hold. In order to differentiate these cases, the noise was first isolated from the output of the Laplacian mechanism. This is done by subtracting 100 and 101 from the output as it must be from one of those cases. We can then reduce the problem of differentiating cases to that of determining whether or not the resulting noise could've been possibly been drawn from the NumPy Laplacian distribution.

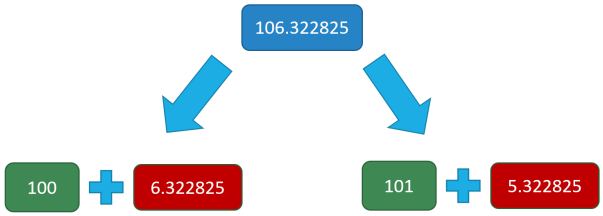

Figure 2: Example diagram of the differentiation problem between two cases of underlying value and noise

A helper function was then developed to answer this question by taking a candidate noise value and returning the candidate uniform random variable associated with it by the quantile function. This was done by first taking the inverse of the quantile function. Because it relies on the sign of the noise, it must be done in two cases. For example, if the noise was positive, then the inverse quantile can be directly calculated to be the following:

$$1 - \frac{1}{2}\exp\left(-\frac{x}{\lambda}\right)$$

Here x represents the Laplacian noise and λ is our noise scale equal to Δ/ϵ. The case for negative noise can be similarly calculated.

Once this inverse quantile value was obtained, it was then quantized to the range of possible multiples of 2^{-53}. It is this quantizing step that uncovers whether the original noise value could've possibly been generated or not by the compute. It was done by running the quantized value back through the inverse cumulative distribution function and comparing it with the original noise. If they do not match then that means the original noise could not have come from the Laplacian distribution as the quantization step rounded the candidate normal distribution value to something else. As a consequence, that case for the underlying true value could not be possible.

Because of other rounding errors that can exist in the process, it was also necessary to check a few values around the quantized variable. However, this was still efficient and does not take much overhead as each one can be computed in constant time. If any of the checks return negative then we can guarantee that the noise value is not possible. However, if all pass then this attack gives us no better information than to randomly guess the underlying value.

Once a test for a single value was developed, it was then scaled up to test a wider range of inputs. Specifically, the privacy budget ϵ was studied to see if it had any impact on the attack success rate. The other parameters such as Δ and underlying values were kept constant. This was done to obtain a better understanding of how the attack performs on a more broad range of cases.

4 RESULTS AND DISCUSSION

After implementing the components described previously in Python and NumPy, the program was successfully able to perform the differentiation test on the Laplace mechanism. In practice, it was necessary to search two quantized values in either direction to have the best chance of detecting an impossible noise value. This simple test shows that while theory tells us that adding Laplacian noise guarantees differential privacy, this is not necessarily the case in practice. The machine precision of floating point values has a non-negligible impact on the algorithm's security.

Moving on to the larger scale test for different values of ϵ, we can see the results graphed in Figure 3. Across the board, the attack success rate remains above 60% and in the majority of cases is even higher than that. It's clear that for a lower λ value the attack succeeds almost all of the time. This is because there is less spread in the noise and the cases are more distinct from each other.

On the far right end of the spectrum, the attack success rate levels out close to 80% which is far from negligible. It is important to note that even when the attack fails, it is still possible to guess the correct case 50% of the time.

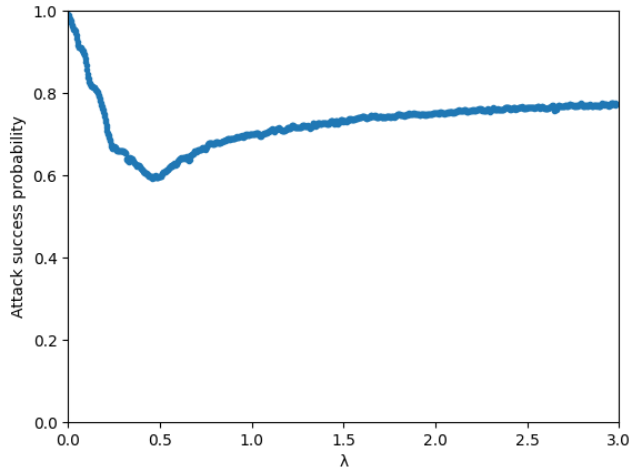

Figure 3: Attack success rates for differentiating between cases with respect to $\lambda = \Delta/\epsilon$

Contrary to what might be expected, however, the smallest values for ϵ which usually implies greater privacy do not always guard against the attack the best. This can likely be attributed to the range and density of possible noise values. With such a small ϵ, the noise magnitude increase as it scales inversely with ϵ. Thus, the range of possible noise values becomes more spread out with more and more gaps that the incorrect case can fall into.

Overall we can see that across all values for our privacy budget, this attack will be able to differentiate between our cases breaking differential privacy.

The interesting section of the graph appears between the two extremes. Within this region, the attack success rate drops to around 60% before recovering and slowly rising to about 80%. This behavior was different from that described by Mironov. In his paper, the graph exhibited a monotonically decreasing behavior instead. The differences may be attributed to the specific intricacies of Python or the library NumPy itself. However, whatever the reason, it still stands that this attack is largely successful across all values of ϵ.

As a last component of this project, a quick survey of existing Python libraries for differential privacy was investigated. Specifically, whether they would be vulnerable to this same precision attack. While some did implement mitigation strategies including rounding or truncating values, there were many that did not [2]. In general, we've shown that simply drawing a Laplacian random variable from NumPy through the use of `np.random.laplace` and adding it to the true value is not secure. However, this naïve approach is still popular because it reflects the standard theoretical form defined in theory. It is clear from this that more awareness needs to be drawn to the nuances in writing custom privacy algorithms as even seemingly small differences can completely break security.

5 CONCLUSION

Much of our computer science has roots in mathematical theory. This is invaluable in providing us with the ability to prove that certain properties will or will not hold. However, we have seen that translating these ideas into practice requires careful consideration. While not always a developer's top concern, random sampling done by computers differs from the ideal theoretical model. While it may not seem significant, this deviation can have dramatic impacts on differential privacy. This project has taught me skills in identifying the quirks that exist from machine floating point representation. It required attention to detail as it focuses on the minute differences between cases. As for the consequences on user privacy, luckily mitigation strategies already exist and can be found in many practical implementations. It is not an issue of guarding against this vulnerability but rather educating developers so they are aware when implementing the Laplacian mechanism.

6 FUTURE WORK

There exists one problem with the inverse Laplace distribution function within my detection algorithm. Specifically, it doesn't always produce the right input uniform random variable likely due to internal rounding by the computer. Currently, the workaround for this issue is by searching in either direction several quanta to check if they are proper inverses.

As mentioned in the prior discussion, this project revealed some interesting behavior for mid-sized ϵ values. It would be interesting to dig further into why the results here differ from that of Mironov. This may uncover some additional details on the inner workings of the language or how floating point numbers are distributed for different magnitude ranges.

Finally, this work focuses on replicating the results of prior researchers. However, there is much more that can be done for this project. One potential avenue is to test the various mitigation strategies that have been suggested and retry the attack to see how effective they are. There could also be a proof-of-concept attack on an actual dataset using one of the publicly available vulnerable implementations of differential privacy. However, the ultimate goal would be to apply the ideas of this attack with floating point precision and develop a novel attack on a different mechanism.

REFERENCES

[1] 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). http://data.europa.eu/eli/reg/2016/679/2016-05-04/eng Legislative Body: OP_DATPRO.

[2] 2023. diff-priv-laplace-python. https://github.com/aleph-research/diff-priv-laplace-python original-date: 2020-03-20T19:53:49Z.

[3] 2023. numpy/numpy. https://github.com/numpy/numpy/blob/7ccf0e08917d27bc0eba34013c1822b00a66ca6d/numpy/random/mtrand/randomkit.c original-date: 2010-09-13T23:02:39Z.

[4] 2023. numpy/numpy. https://github.com/numpy/numpy/blob/c90d7c94fd2077d0beca48fa89a423da2b0bb663/numpy/random/mtrand/distributions.c original-date: 2010-09-13T23:02:39Z.

[5] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography (Lecture Notes in Computer Science)*, Shai Halevi and Tal Rabin (Eds.). Springer, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

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

[7] Jiankai Jin, Eleanor McMurtry, Benjamin I. P. Rubinstein, and Olga Ohrimenko. 2022. Are We There Yet? Timing and Floating-Point Attacks on Differential Privacy Systems. http://arxiv.org/abs/2112.05307 arXiv:2112.05307 [cs].

[8] Ilya Mironov. 2012. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, Raleigh North Carolina USA, 650–661. https://doi.org/10.1145/2382196.2382264

[9] Branislav Stojkovic, Jonathan Woodbridge, Zhihan Fang, Jerry Cai, Andrey Petrov, Sathya Iyer, Daoyu Huang, Patrick Yau, Arvind Sastha Kumar, Hitesh Jawa, and Anamita Guha. 2022. Applied Federated Learning: Architectural Design for Robust and Efficient Learning in Privacy Aware Settings. https://arxiv.org/abs/2206.00807v2