1 Background

The mathematical abstraction of the Laplacian mechanism differs greatly from its implementation in floating point representation. In his paper, *On Significance of the Least Significant Bits For Differential Privacy*, Mironov describes a new type of vulnerability present in implementations of the Laplacian mechanism. Specifically, this vulnerability results from the exploitation of the irregular distribution of the Laplacian mechanism resulting from finite precision and rounding effects of floating point operations.

Let's put this in more concrete terms by using an example of Laplacian sampling. Say we have 10 double numbers, ranging from $x = 1/\pi$ to $x + 9 \cdot 2^{-54}$. Applying the log function to each number in this set of 10 double numbers results in a transformation where some specific output numbers are more frequent than others. Below is some code created to display this exact problem. Notice how two numbers occur twice.

```cpp
#include <iostream>
#include <math.h>

using namespace std;

int main()
{
    cout.precision(20);

    const double pi = 3.1415926535897;
    const double x = 1.0/pi;

    double piNums[10] = {x, x+1.0*pow(2.0,-54.0),x+2.0*pow(2.0,-54.0),x+3.0*pow(2.0,-54.0),x+4.0*pow(2.0,-54.0),
                         x+5.0*pow(2.0,-54.0),x+6.0*pow(2.0,-54.0),x+7.0*pow(2.0,-54.0),x+8.0*pow(2.0,-54.0) +
                         x+9.0*pow(2.0,-54.0)};

    for(int i = 0; i < 10; i++)
    {
        cout << log(piNums[i]) << endl;
    }

    return 0;
}
```

```
-1.1447298858493704099
-1.1447298858493701879
-1.1447298858493701879
-1.1447298858493699658
-1.1447298858493697438
-1.1447298858493695217
-1.1447298858493695217
-1.1447298858493692997
-0.45158270528942356936
-inf
```

Similarly, using a different set of numbers, another thing can happen - the distribution is not evenly distributed and some output values from an interval become missing. This "porous distribution," as Mirinov calls it, is what causes a breach in differential privacy.

To show this in concrete terms, consider the Laplacian mechanism $\tilde{f}(\cdot)_{\epsilon,p} = f(D) + Lap *_p (\Delta/\epsilon)$, where p is the precision with which the uniform distribution over $(0, 1)$ is sampled. Let $\Delta = 1$, $f(D) = 0$, and $f(D') = 1$. Consider the distribution of $\tilde{f}^*(D)_{1/3} = 0 + Lap * (3)$ and $\tilde{f}^*(D')_{1/3} = 1 + Lap * (3)$ around 1.5. As it turns out, the output of the Laplacian mechanism results in some different values for $\tilde{f}^*(D)$ and $\tilde{f}^*(D')$. For example, an output of $\tilde{f}^*(D)$ includes the value 1.5 while $\tilde{f}^*(D')$ does not. Similarly, $\tilde{f}^*(D')$ includes the value $1.5 + 2 \cdot 2^{-52}$, while $\tilde{f}^*(D)$ does not. This means if the output of the Laplacian mechanism is $1.5 + 2 \cdot 2^{-52}$, the input database was D' etc. Thus, differential privacy is violated.

The paper *Fast and Correctly Rounded Logarithms in Double Precision* by Dinechin, Lauder, and Mueller

presents an algorithm to round floating point numbers precisely, to preserve differential privacy. As seen above, there are issues distributing noise with floating point values as they are not precise and can expose data. This new algorithm can provide accuracy up to 2^{-150} if needed. The main structure of this algorithm is comprised of 2 steps. The first step provides accuracy up to 2^{-60}. It executes some operation on the float and then performs a rounding test. If the rounding test passes, this means there is no error and the algorithm can terminate with accuracy 2^{-60}. There is a small chance, however, that there are inconsistencies in the values. In this case, the second step is implemented. It uses predefined libraries based on processor type to provide more accuracy. The output of this step will then be converted to a double precision number and is guaranteed to have accuracy of at least 2^{-150}. By determining a way to integrate this algorithm into the snapping mechanism, we can preserve differential privacy with floating point numbers.

2 Objectives

Mironov proposed something called the *snapping mechanism* to yield a differentially private mechanism by solving the issue explained above. It works by tossing out the lower bits of the Laplacian noise result *after* adding the noise to the query. Previously, our differentially private mechanism was defined as $\tilde{f}(D) = f(D) + Lap *_p (\Delta/\epsilon)$. Using Mironov's snapping mechanism, we now define it to be

$$\tilde{f}(D) \triangleq clamp_B(\lfloor clamp_B(f(D)) \oplus S \otimes \lambda \otimes LN(U^*) \rceil_\Lambda) \tag{2.1}$$

- U^* is the uniform distribution over $\mathbb{D} \cap (0, 1)$ such that each double number is output with probability proportional to its ulp
- S is uniform over $-1, +1$
- $LN(\cdot)$ denotes a floating point implementation of the natural logarithm with exact rounding
- Function $clamp_B(x)$ outputs: (1) B if $x > B$, (2)$-B$ if $x < -B$, (3)x otherwise
- Λ is the smallest power of 2 (including negative powers) $\geq \Lambda$
- $\lfloor \cdot \rceil_\Lambda$ round to the closest multiple of Λ in $\mathbb{D}$ with ties resolved towards $+\infty$

The primary goal of this project is to create a software tool which generates Laplacian noise in a differentially private way by using the above mechanism defined by Mironov. Additionally, we hope to be able to integrate this tool into a larger software framework that is used for querying databases.

3 Deliverables

1. **DP Laplacian Noise Generator using Rust Language**: The goal here is to create a program that can use floating point numbers and generate Laplacian Noise in a way that preserves differential privacy. Rust will be used since it is a secure language that provides all the necessary tools we may need to complete the algorithm.
2. **Integrated Software**: Once we have the basic algorithm completed, the next step is to determine what pre-existing frameworks we could potentially integrate the tool with. The purpose of this would be to ensure differential privacy with large database systems. After researching what possibilities exist, we would then modify our code so that the tool could work hand in hand with the framework.

3. **Software Documentation**: Next we need to provide adequate documentation for our code. The goal would be for any user to be able to read and understand the code so they could use it with a database they may have. The documentation would include not only in code comments, but also a guide to explain how it works.

4. **Software Bindings for C/Python (optional)**: Many projects related to this work are in C/Python. To make our code more accessible and flexible, we could add bindings so that it will work no matter which language it is to be used with.

5. **GUI for Software Tool (optional)**: If time allows, we could also create a GUI to make our tool more user friendly. This way, when users have a database, they would not need many software skills/terminal knowledge. The GUI would guide them through the process of adding noise to their database.

4 Project Timeline

1. **DP Laplacian Noise Generator using Rust Language**: October 16th

2. **Integrated Software**:November 17th

3. **Software Documentation**:November 30th

4. **Presentation**: November 30th

5. **Software Bindings for C/Python (optional)**: If time allows

6. **GUI for Software Tool (optional)**: If time allows

References

[dLM07] de Dinechin, Florent, Lauter, Christoph, and Muller, Jean-Michel. Fast and correctly rounded logarithms in double-precision. *RAIRO-Theor. Inf. Appl.*, 41(1):85–102, 2007.

[Mir12] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 650–661, New York, NY, USA, 2012. ACM.

[Mir12] [dLM07]