How to break, then fix, differential privacy on finite computers

Or: what do you do when $x + y = \text{privacy vulnerability}$?

Damien Desfontaines
damien@desfontain.es
@TedTed@hachyderm.io

Samuel Haney
sam.haney@tmlt.io
**Differential privacy in one slide**

**Differential privacy**: the impact of a single person must be **undetectable**.

Counting the number of records: the true answer is either 100 or 101.
What happens to our continuous line?
Why does this happen?

def add_noise(true_value, epsilon):
    sign = random.choice([-1, 1])
    u = random.uniform(0, 1)
    noise = sign * math.log(u) / epsilon
    return true_value + noise

This does not generate all possible floating-point values between 0 and 1!

This creates “holes” — impossible values — in the noise distribution...

And the “holes” propagate to the sum.
def add_noise(true_value, epsilon):
    sign = random.choice([-1, 1])
    u = random.uniform(0, 1)
    noise = sign * math.log(u) / epsilon
    return true_value + noise

Let’s fix the noise generation!

Attempt 1: fixing the noise generation to get a distribution without “holes”.

Attempt 2: combining multiple noise samples together to make it intractable to reverse-engineer the randomness.

But… what about the sum at the very end?
Fun fact about floating-point addition...

- precision: $2^{-52}$
- precision: $2^{-53}$
- precision: $2^{-54}$ and below
- precision: $2^{-53}$
- precision: $2^{-52}$
Fun fact about floating-point addition...

What if we add noise to 1.25? It has precision $2^{-52}$.
Fun fact about floating-point addition...

If the noise is small...
Fun fact about floating-point addition...

If the noise is small...
the sum’s precision is at least $2^{-53}$. 
Fun fact about floating-point addition...

If the noise is large...
Fun fact about floating-point addition...

If the noise is large...
the sum is a multiple of $2^{-53}$!
Takeaway: this is bad news

When adding noise to a number of precision $2^k$, we always get a multiple of $2^{k-1}$.

true value: 1.25

true value: 0
How do we fix it?

def add_noise(true_value, epsilon):
    sign = random.choice([-1, 1])
    u = random.uniform(0, 1)
    noise = sign * math.log(u) / epsilon
    return true_value + noise

We need to fix the entire routine, not just the noise generation!
General aim

Generate the distribution centered on the true value

Use the inverse of the cumulative distribution function
Sample intervals instead
Interval refining
Interval refining
Interval refining
Rounding the interval

64-bit floating-point values
Termination condition
One more detail... interval arithmetic
Why this is neat

- **Simple security proof**: “just like” infinite-precision sampling + rounding!💡

- **Fully generic**: works with many distributions, adapts to other methods!✨

- **Fast**: converges quickly, especially if we generate many bits at a time 🏎️
Takeaways

- Differential privacy can have vulnerabilities! 😱

- To fix them, ad hoc approaches are not robust enough 🚫

- But principled approaches can be simple (and fast) enough! 🎉

- What do you need to do? Nothing — just use a library with a proven fix 😊
## Impact & mitigations

<table>
<thead>
<tr>
<th>Library</th>
<th>Status</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartNoise Core</td>
<td>Vulnerable, won’t fix</td>
<td>Project was deprecated</td>
</tr>
<tr>
<td>Diffprivlib</td>
<td>Vulnerable, not fixed</td>
<td>Snapping mechanism available for Laplace, no fixes for other distributions</td>
</tr>
<tr>
<td>OpenDP</td>
<td>Vulnerable, then fixed</td>
<td>Configurable discretization parameter, doesn’t generalize</td>
</tr>
<tr>
<td>GoogleDP</td>
<td>Not vulnerable</td>
<td>Fixed discretization parameter, small privacy cost, doesn’t generalize</td>
</tr>
<tr>
<td>Tumult Analytics</td>
<td>Not vulnerable</td>
<td>Hyperparameter-free, generalizes ✨</td>
</tr>
</tbody>
</table>

Thanks to everyone who ships open-source code 💚
Stay in touch!
We’re Sam Haney and Damien Desfontaines on the PEPR Slack 🌶

Learn more!
About us: tmlt.io
About our code: tmlt.dev

Thank you ♥