Differential Privacy from the Ground Up, or; Differential Privacy for Dummies

By[1]: Joseph “Joey” Knightbrook (knightbrook@google.com)
15 April, 2022

Pre-Prelude

Prelude

What’s the problem?

Counting Queries

Noisy Counting
- Gaussian Noise
- Laplace Noise
- Formal Definition

Whither Epsilon? Finding a Reasonable Value

Noise and Epsilon Relationship: A General Relationship through Sensitivity Analysis
- Sensitivity Examples
  - Counting Users
  - Count Impressions
  - Min/Max of a number

Handling Revenue-Critical Numbers: The Exponential Mechanism

Composing Queries: Epsilons Add together

Privacy Budget: The Final Frontier

Different Definitions of “Neighbor”
- Protect Users
- Protect Records
- Protect Partitions

Appendices
- A Bayesian Aside
  - Where ln(3) at Google Actually Comes From
- Additional Reading

[1] This doc has been used by over 2000 Googlers to learn DP at this point, many of whom have added very helpful comments and who I am indebted to for this intro’s current state and quality.

Pre-Prelude

Differential Privacy protects user data by adding statistical noise to a “data release” to confuse attempts by an attacker to learn information about individuals. Intrigued!? Please read on…

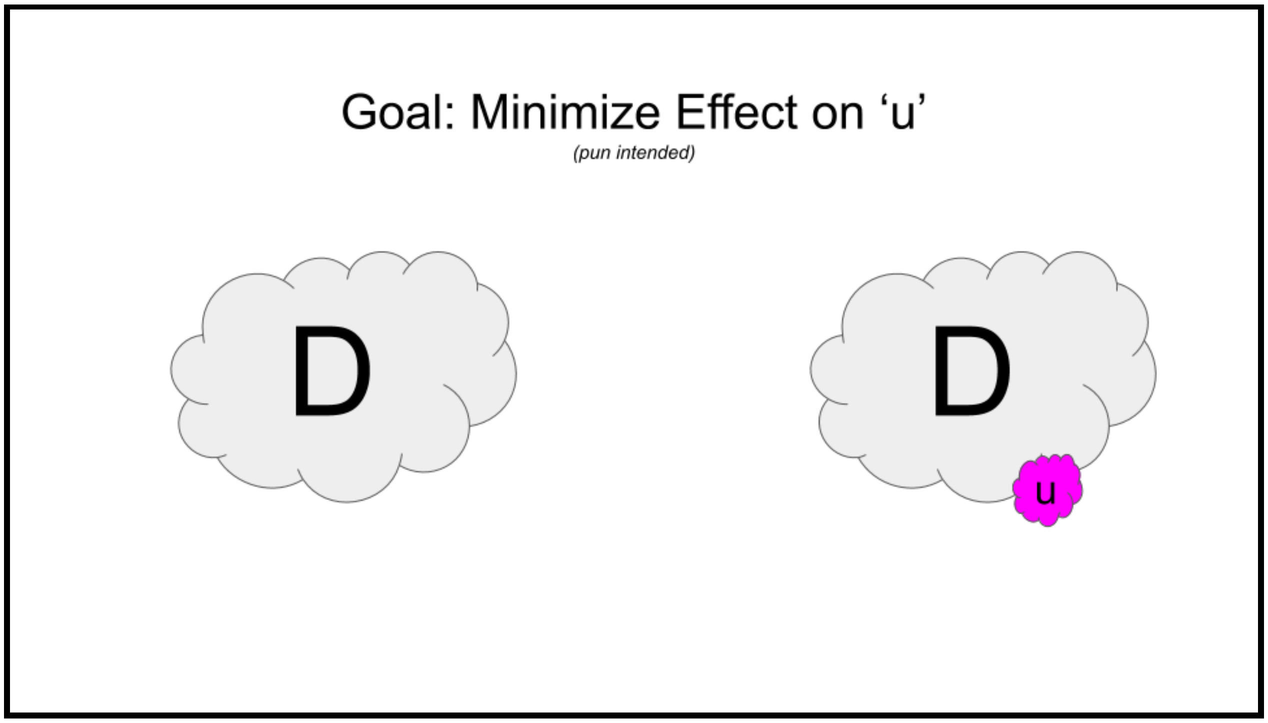
Prelude

Lots of “introductions” to differential privacy, I find are lacking in motivation, and plop down the definition without explaining a lot of weird things in the definition. Take “The Book” on differential privacy, by Dwork and Roth. Great Mathematical introduction, which I love and am using as inspiration for this doc, but they assume you understand the motivation and just plop you right into it, and don’t really derive things from first principles. So this is my attempt at building up from nothing the definition for Differential Privacy.

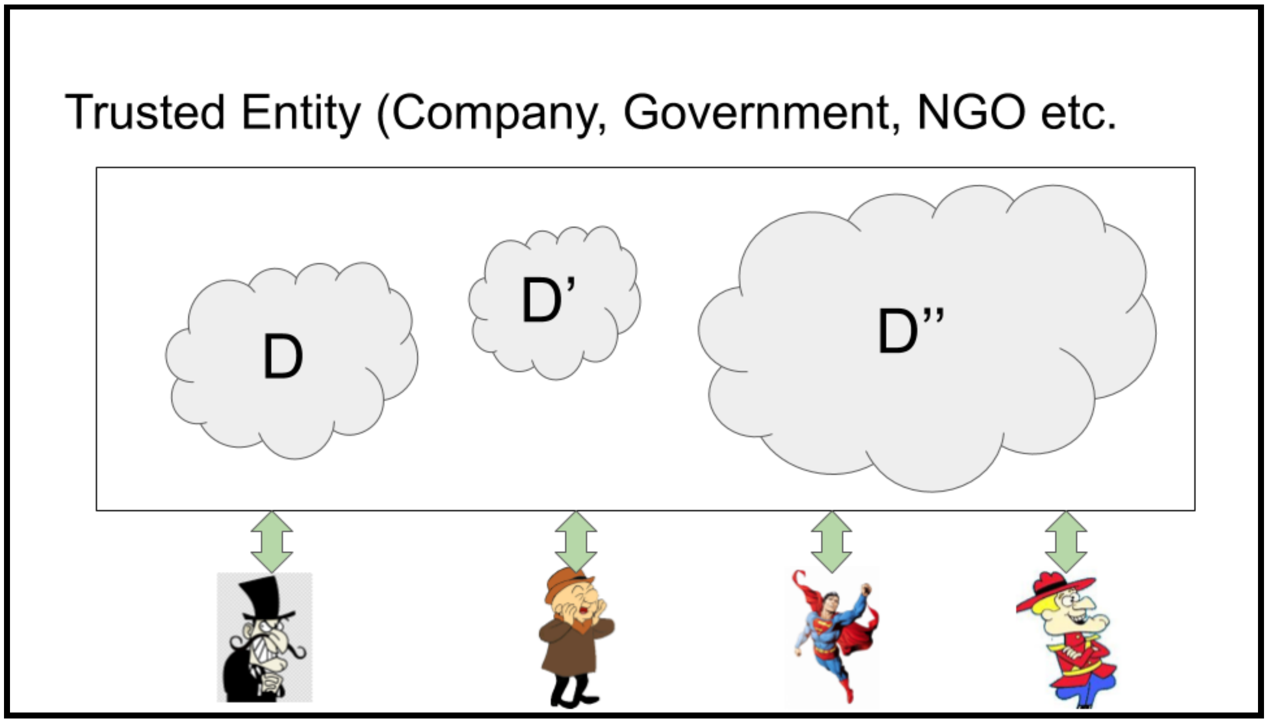
What’s the problem?

If you are the owner/curator of a set of data, you want to make your data useful to people, and be able to make useful conclusions about groups of people. A great example of this might be a study on smoking. You want to be able to make inferences about groups of people, as in what behaviors make cancer more likely for smokers, but you don’t want the identities of smokers to leak. Otherwise, these smokers wouldn’t want to be in your study, since they might not want people to know they are smokers, or have their medical records potentially linked to them personally and have their healthcare costs raised.

With this goal in mind, how can we begin to build up a mathematical formalism that protects one user, but still allows inferences to be made about the group?



Also implicit in this model of privacy is the fact that you have a trusted curator of data like a company, government, or NGO upon which people are asking to learn information about:

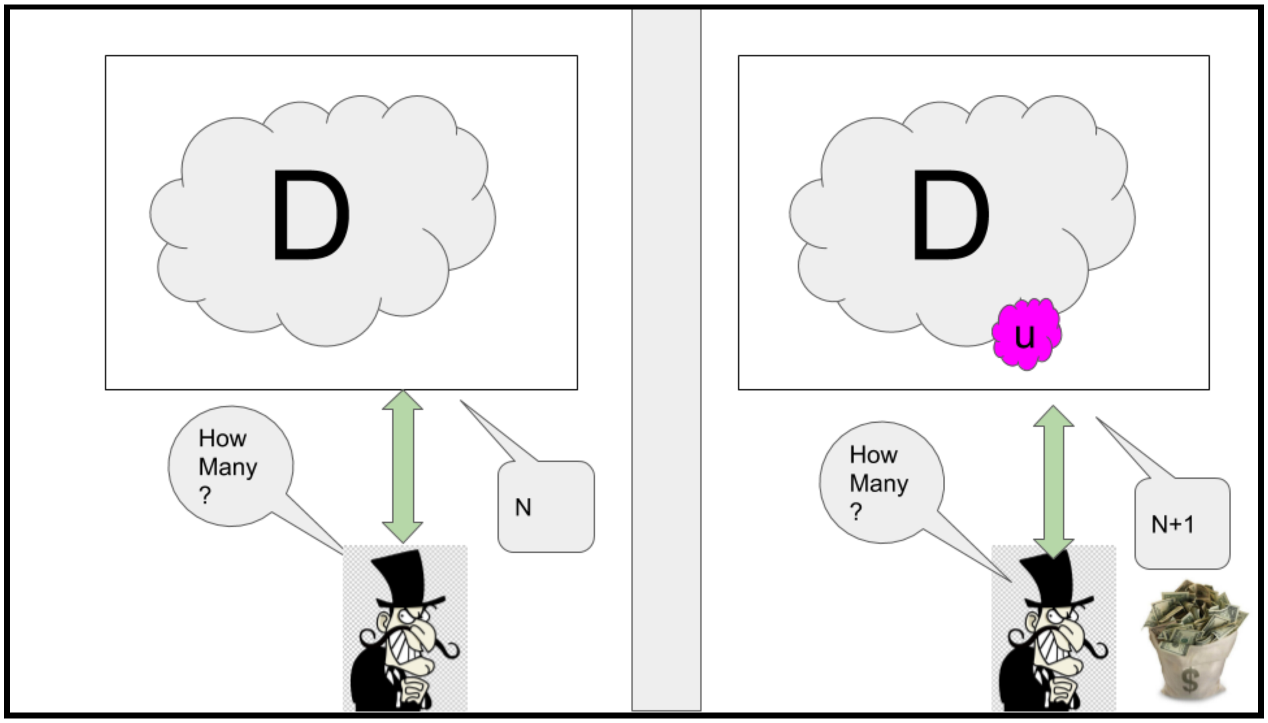


But you don’t know who the people are that are trying to learn this information: they could be nefarious, incompetent, or want to help, or want to help but also be incompetent. So you want to be able to thwart the nefarious ones and minimize the damage an incompetent person can do, but not get in the way of those who want to do good. The answer to all of these is just to make sure that no one can get too much information about a single person.

So how do we do this!? Let's start with an example of the most basic failure of this kind.

Counting Queries

Let's imagine the situation where a nefarious attacker just asks a database the question: how many people are in this database?



There is a failure here 100% of the time and u gets harmed! If the attacker is able to ask the question first of the database without u in it, then ask the database the same question knowing u is now in the database: they are able to get that information out, and thus exploit their knowledge. This might seem far fetched, but imagine the case where you want to know if someone is in a specific location so you can rob the person's house.

This might also seem like a real contrived example of just asking how many people are in one single database, but this applies for subsets of a larger database too! Any query one could make where you are counting things in a database, there is a problem. And really, a lot of queries to databases are just as mundane as counting how many people/rows fit a certain definition.

Is this too strict? The issue is that with privacy (like with security) you worry about the absolute worst-case scenario, as that's where the real harm to users occurs. So in a worst case scenario, Snidley has a lock on the database so that no one else can make modifications to the data. **So the attacker can know they made a change to the database and get out that information.** Implicit in this assumption is also that the attacker has some background knowledge of the user. For example, let's say this counting query is for hospital admissions on a certain day from a certain zipcode, and the attacker sees their neighbor being taken away in an ambulance. They could perform the select disease, count(*) from hospital_admissions before and after the ambulance goes away to get information about what their neighbor had.

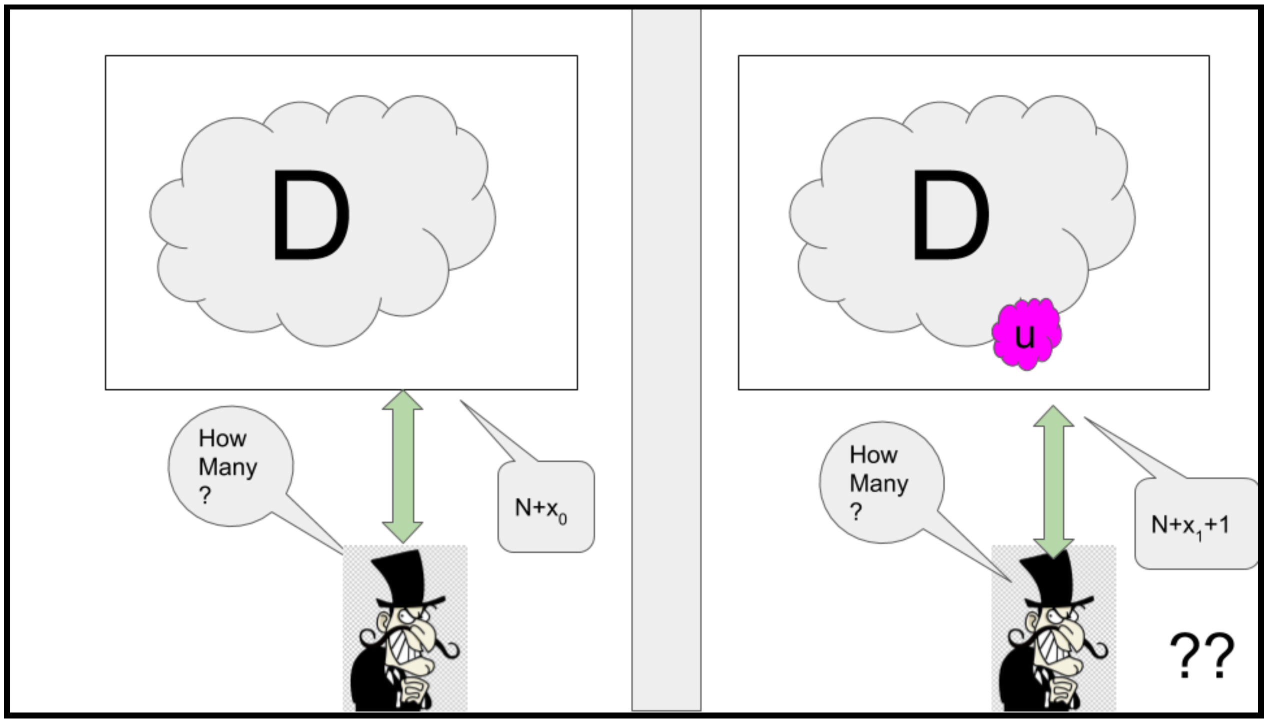
So how can we possibly confuse the nefarious Snidely Whiplash so he can't take u's money while they are sick and in the hospital!? One might think of the thing that causes numerical confusion in science experiments: noise.

Noisy Counting

What if instead of just returning the count, we return the count plus noise drawn from some noise distribution: crucially choosing a different noise every time someone asks the database a question? Let's define

$$x_i \sim \rho(\sigma)$$

Where x_i is one of the series of drawn random numbers, and ρ is a generic probability distribution (since ρ looks like p for probability distribution) with some scale parameter sigma. So what does our previous situation look like now?



And now, depending on how much noise is added, Snidely is some amount of confused as to whether u is actually in the database or not: cool!

But how much protection is that: **how confused is Snidley?** Let's introduce some notation to make things easier to talk about and write math about.

First let's call this algorithm where we count, then add a new piece of noise, M: a randomized algorithm.

$$M(D) = \text{Count}(D) + x_i$$

Then let's talk about the value measured as being ⌊ (lower-case L). And let's consider the probability that we measure the ⌊ in both scenarios, first without u in the database:

$$P\left[M\left(D\right) = l\right]$$

Versus, with u in the database:

$$P\left[M\left(D + u\right) = l\right]$$

So the probability that we got the specific measured value given that u is in the database versus the other universe where u is not in the database. Why not take the ratio and see how that changes?

$$r = \frac{P\left[M\left(D + u\right) = l\right]}{P\left[M\left(D\right) = l\right]}$$

And this should be calculable! The ratio that you get an observation ⌊ given there are N+1 people in the database versus the probability that you get the same observation with only N people.

And this should definitely get at the question of just how confused the nefarious Snidley should be:

If the ratio is big: that's bad! You have learned "more" because the one likelihood is much more likely in the N+1 case.

If the ratio is too small: that's bad! You have again learned that one answer is much more likely in the N case. So we want to bound this ratio from above and below. This is already suggesting we might want to take the log of r since we know it will be above zero, but we'll get back to that later.

If, however, the closer this ratio is to one, the more confused Snidley should be, because that's when he has much less of a clue as to which database he's looking at.

Gaussian Noise

We can't go any further without choosing a noise function so let's choose Gaussian Noise since that seems to be the cool noise everyone likes to pick for everything. You can imagine the gaussian/normal distribution is centered at the "true" count of people in the database, the standard deviation is the scale parameter, and our observation point in the probability distribution function (pdf) x is what we observe from the count,

$$\begin{aligned} r &= \frac{P\left[M\left(D+u\right)=l\right]}{P\left[M\left(D\right)=l\right]} \\ &= \frac{\text{norm\_pdf}\left(\mu=N+1,\sigma=\sigma,x=l\right)}{\text{norm\_pdf}\left(\mu=N,\sigma=\sigma,x=l\right)} \\ &= \frac{e^{-\left(l-\left(N+1\right)\right)^2/2\sigma^2}}{e^{-\left(l-N\right)^2/2\sigma^2}} \\ &= e^{\left[2\left(N-l\right)+1\right]/2\sigma^2} \end{aligned}$$

Alright cool, we have something derived and you might even get excited since there are values of N and l for which r=1! But unfortunately, this expression is not bounded: if N is too big in relation to l, r is huge, and if l is too big in relation to N, then r is too small. So this is promising but it has problems.

Laplace Noise

Really the thing which gave us problems with Gaussian noise was the square in the power of e: getting rid of that seems promising. So if you look through your catalog of probability distributions, you'll find that Laplace fits the bill:

$$\text{laplace\_pdf}(x|\mu,\sigma) = \frac{1}{2\sigma}e^{-\frac{|x-\mu|}{\sigma}}$$

So now what does r look like?

$$
\begin{aligned}
r &= \frac{P\left[M\left(D+u\right)=l\right]}{P\left[M\left(D\right)=l\right]} \\
&= \frac{\text{laplace\_pdf}\left(\mu=N+1,\sigma=\sigma,x=l\right)}{\text{laplace\_pdf}\left(\mu=N,\sigma=\sigma,x=l\right)} \\
&= \frac{e^{-|l-(N+1)|/\sigma}}{e^{-|l-N|/\sigma}} \\
&= e^{-[|l-(N+1)|-|l-N|]/\sigma}
\end{aligned}
$$

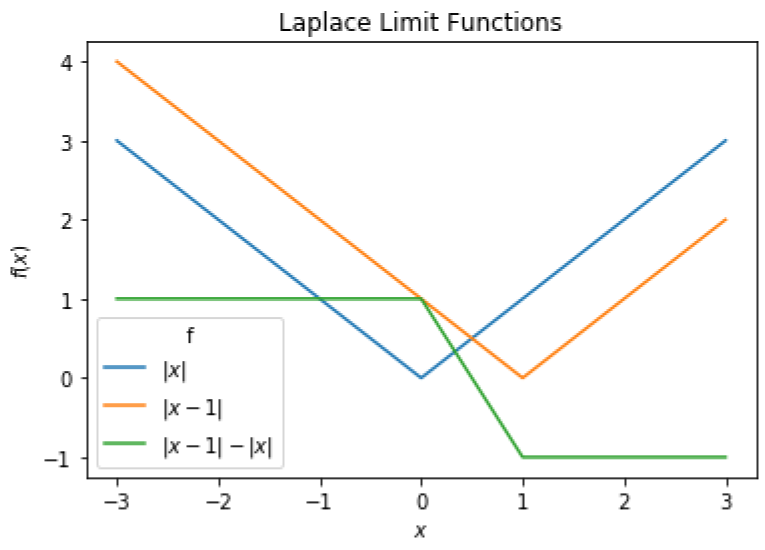
Now we are getting somewhere! The difference of those absolute value terms is most certainly bounded. A fact we can more easily see if we move things around a bit and do a substitution for l-N into the ubiquitous cartesian x dimension:

$$r = e^{-[|(l-N)-1|-|(l-N)|]/\sigma}$$

$$(l-N) \rightarrow x$$

$$r = e^{-[|x-1|-|x|]/\sigma}$$

Since absolute values can be a little hard to reason about, I'm going to draw a graph in x:



Where you can clearly see the magnitude of this difference is going to stay between 1 and negative 1: which is exactly what we need for some kind of information release guarantee! So we can assign the inequality:

$$e^{-1/\sigma} \leq r \leq e^{1/\sigma}$$

Now this is more like it! This has nice bounds that don't depend on the size of the problem. If you're willing to add infinite noise, you could even get perfect protection… but you don't want to do that, do you?

Formal Definition

We got lucky for the counting query using Laplacian noise, that there was no dependence on the output of the algorithm M, and the actual parameters, and we're going to be pessimistic because this is privacy and it's important. And there could be cases where u is a special person (of course u is special), so we want this ratio to hold for any, let's call them "neighboring" databases D and D'. Because depending on your desired protection/definition of neighbor, you might want to consider the act of making a neighbor database, adding or deleting the thing you want to protect against, which might not be a user: it could be an event. As an example, with ads data, you might want to protect a user's history (so single impressions or clicks), or you might want to protect a holistic user. So we keep "neighbor" as being an abstract concept, and it's up to you to decide how your application needs to define neighbor. And we will now introduce the limiting parameter as epsilon.

And because we're talking about ratios, and we want to bound from zero, we will now take that logarithm. All told, our definition is now:

$$-\epsilon \leq \ln\left[\frac{P[M(D) = z]}{P[M(D') = z]}\right] \leq \epsilon$$

FOR ANY POSSIBLE OUTPUT z, and for ANY NEIGHBORING DATABASES D and D', we want to be able to get a strict bound on the log of the ratio of their probabilities. **A randomized algorithm M is said to be ϵ-differentially private if it obeys this ratio.**

Now you might have an algorithm which only obeys this ratio some of the time, and we will run into that problem in future sections. If your algorithm fails this inequality a ratio δ of the time, it is said to be (ϵ, δ) differentially private, where this inequality holds 1-δ of the time: δ is the ratio of times this inequality fails.

Now some of you, I've heard, have seen this beautifully symmetric equation defined in this less symmetric way:

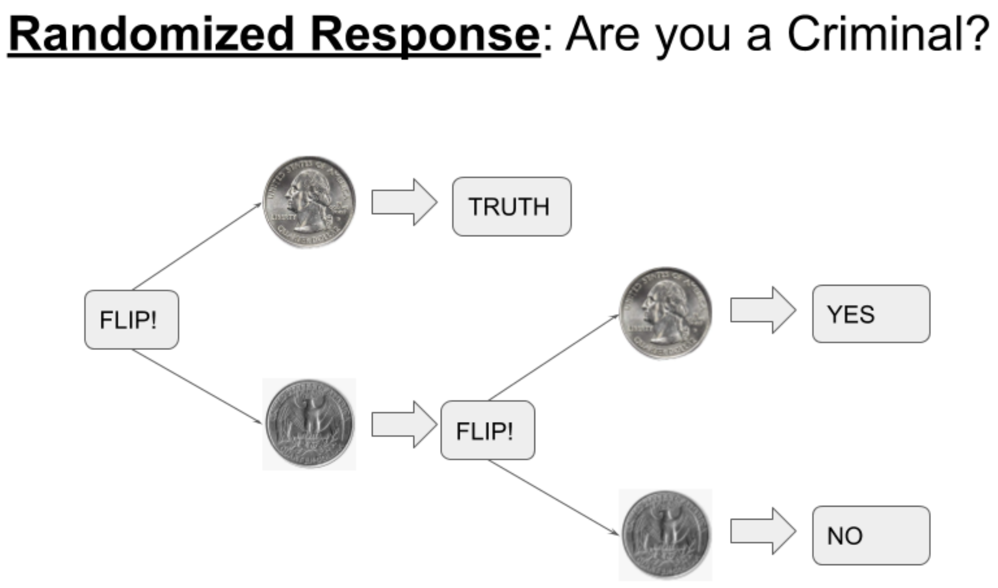
$$P[M(D) = l] \leq e^{\epsilon} P[M(D') = l] + \delta$$

And these are definitely equivalent! I just think it's easier to motivate the way I wrote it first, and the symmetry is clearer. You WILL see the above way basically everywhere else, though, so be warned. ...But if you like my way of writing it: keep putting it out there, please!

Whither Epsilon? Finding a Reasonable Value

What is a good value for ϵ? It seems rather arbitrary what the value should be. Where do we even begin? What might be a good waypoint would be some process that we know people are generally comfortable doing when telling their data to a researcher. For that reason we look to the technique of randomized response from survey design.

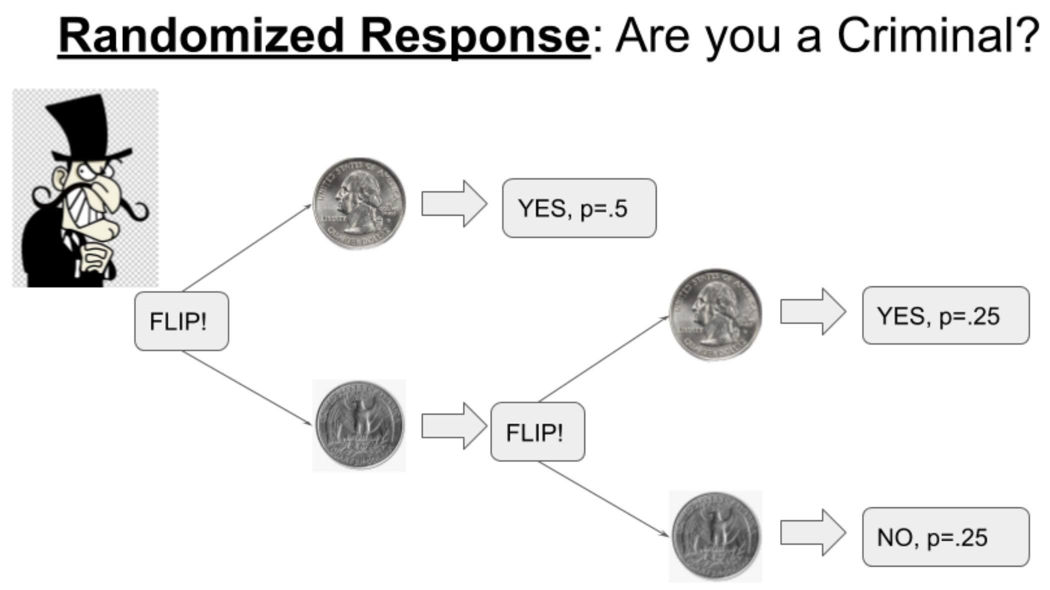
The Idea behind randomized response is to inject plausible deniability into the asking of a sensitive or embarrassing question. One way of doing this that we know people are more comfortable with responding correctly to [CITATION NEEDED] is thus:



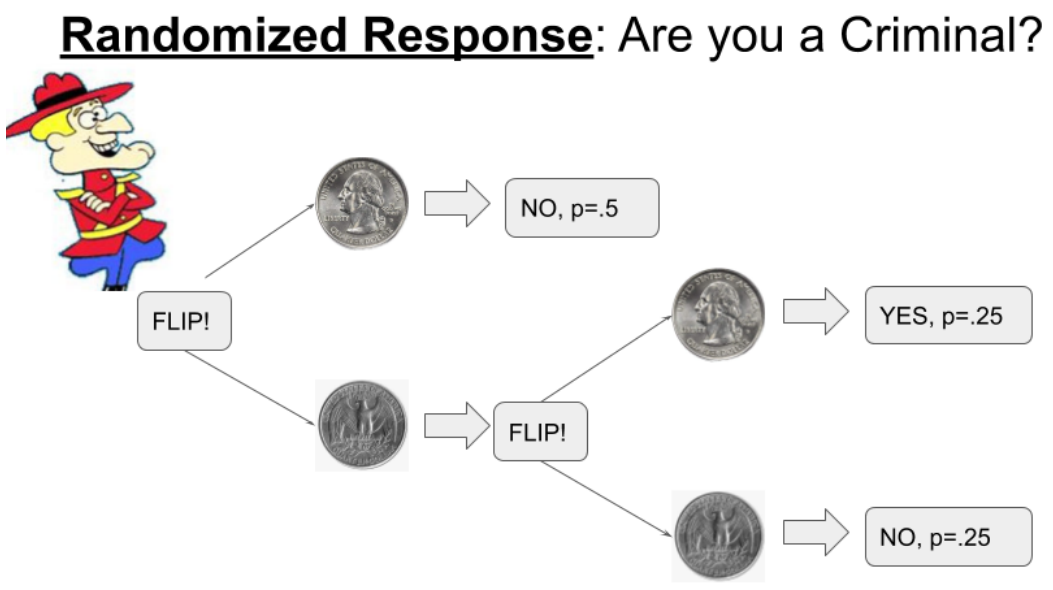
Ask them to flip a coin: if it is heads, answer truthfully, otherwise flip another coin to determine whether you answer heads or tails. It turns out that people are actually generally comfortable answering this random question truthfully to researchers, and with minimal assumptions, you can look at this in terms of differential privacy.

The main obstacle is co-opting the language of neighboring databases: if we assume the case where one participant is a criminal versus the case where one person is not a criminal are neighbors, then we can define an ϵ for this randomized response algorithm, once again calling it M.

Starting with the criminal case:



There is a total probability of .75 that the observation z=YES will occur, and a total probability .25 that z=NO will occur. Then, for the non-criminal case:



There is a probability of .75 that z=NO is observed, then a probability of .25 that z=yes is observed. So given there are two possible outcomes, we need to investigate both ratios:

$$
\begin{aligned}
r_1 &= \frac{P[M(\text{criminal}) = \text{YES}]}{P[M(\text{not-criminal}) = \text{YES}]} = \frac{.75}{.25} = 3 \\
r_2 &= \frac{P[M(\text{criminal}) = \text{NO}]}{P[M(\text{not-criminal}) = \text{NO}]} = \frac{.25}{.75} = \frac{1}{3} \\
\epsilon &= \ln(3)
\end{aligned}
$$

Which has a nice symmetry to it, straightforwardly getting us to a semi-reasonable value of ϵ, being ln(3). You might decide your information is less sensitive and assign a higher value of ϵ to protect it.

Noise and Epsilon Relationship: A General Relationship through Sensitivity Analysis

Not every randomized algorithm will be as simple to work with as the counting algorithm. For that reason, we really ought to develop a formalism that can work with any query on any database. In this section we will attempt to develop that kind of algorithm.

For starters, not every query will just be a single number: some will be multiple numbers. So let's turn our randomized algorithm M on a database x, into a multidimensional vector, and separate it out into its deterministic part f, and the random part Y which will be laplace noise of scale sigma:

$$
\vec{M}(x) = \vec{f}(x) + \vec{Y}(\sigma) = \vec{z}
$$
$$
M(x)_i = f(x)_i + Y(\sigma)_i = z_i
$$

Where the subscript i is just the vector index, and z continues to be the observations.

Because we are now calculating the overlap of many probabilities: the probability of z0 AND z1 AND … We have to multiply probabilities for both the neighboring databases x, and y, and I will introduce a shorthand $p_x(z)$ where x becomes the database state, and z the observed output, and the random mechanism M is implied and the same for all. Putting that altogether like so:

$$
\begin{aligned}
r = \frac{p_x(\vec{z})}{p_y(\vec{z})} &= \prod_i \frac{e^{-|f(x)_i - z_i|/\sigma}}{e^{-|f(y)_i - z_i|/\sigma}} \\
&= \prod_i e^{-(|f(x)_i - z_i| - |f(y)_i - z_i|)/\sigma}
\end{aligned}
$$

At this point, when subtracting absolute values of things, you might think you could use the triangle inequality (or more accurately, its close cousin the reverse triangle inequality) to continue with this derivation, and you would be right:

$$r = \prod_i e^{-(|f(x)_i - z_i| - |f(y)_i - z_i|)/\sigma}$$

$$r = \prod_i e^{(|f(y)_i - z_i| - |f(x)_i - z_i|)/\sigma}$$

$$r \leq \prod_i e^{|f(y)_i - f(x)_i|/\sigma}$$

Alright, once again this is cool, but how can we go further? Let's tuck the product into the sum like so:

$$r \leq e^{\frac{1}{\sigma} \sum_i |f(y)_i - f(x)_i|}$$

And now we have what amounts to the L1 norm of the difference between the two vectors of the deterministic parts of the algorithm which seems to be knowledge we might have beforehand: the maximum amount a function of the database changes when you add a user/row to it. Again practicing pessimism, we take the maximum possible value for this difference (for ANY two neighboring databases, and define it as the "sensitivity" Δf:

$$\Delta f = \max_{\text{Neighbors}(x,y)} \sum_i |f(y)_i - f(x)_i|$$

$$r \leq e^{\Delta f / \sigma}$$

$$\epsilon = \frac{\Delta f}{\sigma}$$

So there you have it: a general worst-case-ϵ guarantee for adding Laplace noise to a vector-value function of a database. While cool, this is still just a bound: for certain algorithms you could do better than the inequalities we introduced, and thus get a tighter value for ϵ, and thus a smaller amount of noise added to your algorithm, and thus better research inferences made for the same amount of desired user-protection.

Sensitivity Examples

Counting Users

When a user can only appear in 1 vector entry, the sensitivity is 1. If a user could conceivably appear in more than one vector entry, then it is more complicated.

Count Impressions

This is tricky from the start. If your definition of neighboring is just adding a impression/row to the database, then the answer is the same as above.

If, however, you care about protecting USERS, not just impressions, then you start to have a problem. If your database mechanism doesn't filter the number of impressions a single user can have, then the sensitivity strictly speaking is infinity because a new user could add an infinite amount of impressions: so not possible to get any differential privacy bound on user-level protections.

This maximum contribution filtering is a concept which will keep coming up: you have to limit the contribution a single user can make, to guarantee user-level differential privacy.

Min/Max of a number

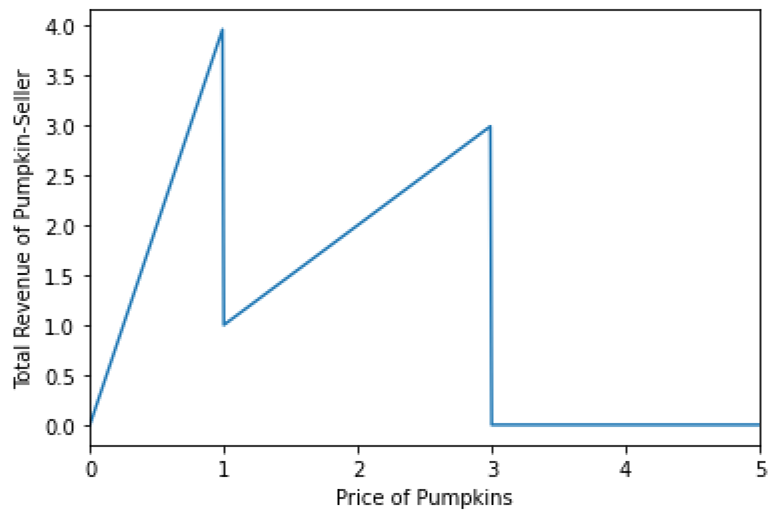
This is hard in all cases: the sensitivity of returning the max or the min is again, always infinity, unless you set a maximum and minimum number for what you would return. If you want a DP max/min you have to get clever, and there are many ways to be clever.

Handling Revenue-Critical Numbers: The Exponential Mechanism

You might imagine a situation where you REALLY need the noise to end up on one side of a number instead of another: for example in an auction situation. Say you are running an auction for pumpkins, and you want to know what price you should set, but in a differentially private manner, so no pumpkin-buyer gets too much information about any of the other pumpkin buyers. Taking the example from Dwork section 3.4, let's assume you have 4 bidders:

Bidder	**Price($)**
A	1
F	1
I	1
K	3

So if you set your price at 3.01 or 1.01, you stand to lose a lot of money, versus setting the price at 2.99, or 0.99 like this graph shows of your total revenue:



So how can we differentially-privately return a price for our precious pumpkins!?

The answer lies in a trick someone very clever (McSherry and Talwar) came up with what is called the exponential mechanism. You'll note in the graph above, we have essentially come up with a utility function: a function of both the database, and a potential price which tells us how "happy" we will be with that price. If you're not familiar with the economic concept of utility, it's a wrapping up of money, preferences, happiness etc. I wrote happy because you can basically think of it as an extremely abstract number of how happy you would be with a specific outcome. Here we'll define it as just straight money, but you can also potentially if, for example, you REALLY value some money over zero, you could instead take e to the power of the money you get and set that as your utility function. There are lots of different manipulations one could make.

The claim of the exponential mechanism, is if you define a utility function **u** over the database x and the output r, and sample from the exponential of that utility function, you can get an ϵ-Differentially-Private algorithm.

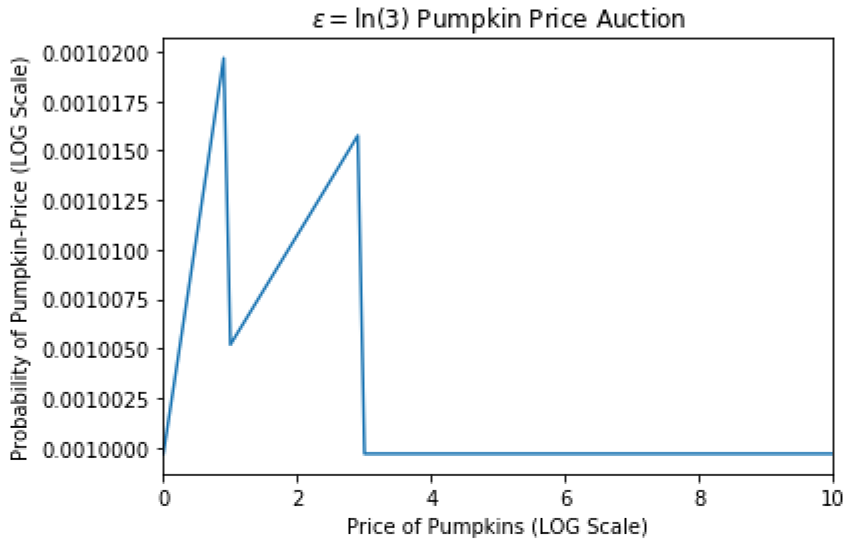
$$r_{\text{output}} \sim e^{\frac{\epsilon}{2\Delta u} u(x,r)}$$

$$\Delta u = \max_{\text{Neighbors}(x,y),r} |u(x,r) - u(y,r)|$$

Where we had to extend the definition of sensitivity for u to include the output price. And the factor of 2 will get explained soon: promise!

Figuring out the sensitivity is once again tricky, however. In the pumpkin-auction example, it's the effect one new bidder's bid could have on the price output which again without filtering would be infinity: you could imagine if Bill Gates REALLY wants a pumpkin, he could bid x where x >> 3, and the utility would go from 0 to x. So unless you limit the max x, you cannot get differential privacy.

Placing a limit in an auction sense does make quite a bit of sense: the auction house can assume grotesque bids aren't serious, and just drop them. So if we assume that's a \$100/pumpkin limit, we can plot the probability distribution you would be sampling from for ϵ=ln(3):



Ok, you might be saying, that's cool and I could write an algorithm to sample from the distribution, but you still haven't proven anything! Alright fine, I'll do it now.

To get into the language of differential privacy, we have to now extend our definition of a randomized algorithm M to be over the utility function as well, and we'll assume a specific domain of the price to be R: imagine this is all the cent-increments between 0 and the maximum bid of 100:

$$P[M(x,u,R)=r] = \frac{e^{\epsilon u(x,r)/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(x,r')/2\Delta u}} = \frac{1}{N_x} e^{\epsilon u(x,r)/2\Delta u}$$

So the probability of getting the specific price p is the exponential at that point p, divided by the sum of the exponentials at ALL possible points p'. Now we look at the neighboring database y and take the ratio again:

$$p_y = P[M(y,u,R)=r] = \frac{1}{N_y} e^{\epsilon u(y,p)/2\Delta u}$$

$$\begin{aligned}
\frac{p_x}{p_y} &= \frac{N_y}{N_x}\frac{e^{\epsilon u(x,r)/2\Delta u}}{e^{\epsilon u(y,r)/2\Delta u}} \\
&= \frac{N_y}{N_x} e^{\epsilon(u(x,r)-u(y,r))/2\Delta u} \\
&\leq \frac{N_y}{N_x} e^{\epsilon \Delta u/2\Delta u} \\
&= \frac{N_y}{N_x} e^{\epsilon/2}
\end{aligned}$$

Which leaves us with the coefficients:

$$\frac{N_y}{N_x} = \frac{\sum_{r' \in R} e^{\epsilon u(x,r')/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}$$

$$1 = \frac{e^{\epsilon u(y,r')/2\Delta u}}{e^{\epsilon u(y,r')/2\Delta u}}$$

$$\frac{N_y}{N_x} = \frac{\sum_{r' \in R} 1 \times e^{\epsilon u(x,r')/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}$$

$$\frac{N_y}{N_x} = \frac{\sum_{r' \in R} \frac{e^{\epsilon u(y,r')/2\Delta u}}{e^{\epsilon u(y,r')/2\Delta u}} \times e^{\epsilon u(x,r')/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}$$

Which I'll pause briefly here to explain that we have just used the old mathematical trick of multiplying by something that equals one. From that, we will then have the same ratio from the first part of the derivation of r, which we already know to be bounded by the exponential of half of ϵ:

$$\frac{N_y}{N_x} = \frac{\sum_{r' \in R} \frac{e^{\epsilon u(y,r')/2\Delta u}}{e^{\epsilon u(y,r')/2\Delta u}} \times e^{\epsilon u(x,r')/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}$$
$$\leq e^{\epsilon/2} \frac{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}{\sum_{r' \in R} e^{\epsilon u(y,r')/2\Delta u}}$$
$$\frac{N_y}{N_x} \leq e^{\epsilon/2}$$
$$\frac{p_x}{p_y} \leq e^{\epsilon}$$

And thus there is the proof that this exponential mechanism works, and an explanation for why that weird factor of one-half was there: both the normalization constant denominator, and the probability function numerator contribute a factor of 2 each.

There are actually many more places where we will see an addition of ϵs to get a total ϵ: which we will explore in the next section:

Composing Queries: Epsilons Add together

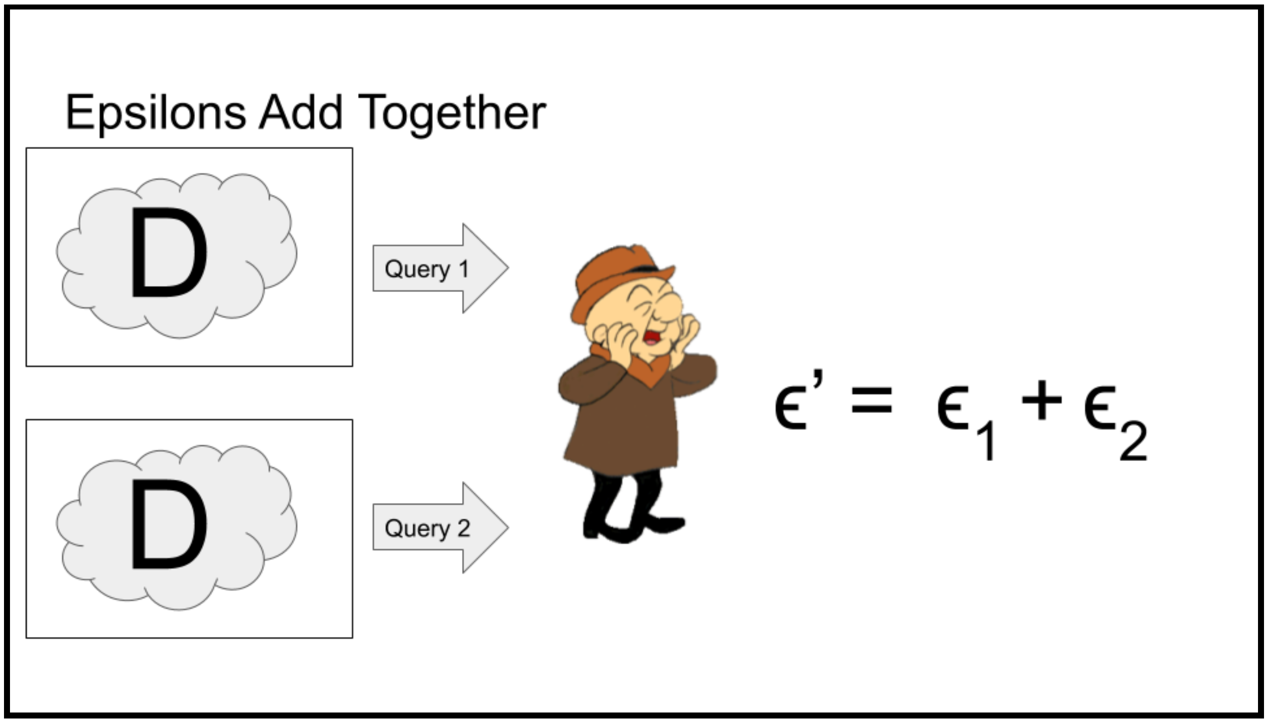
If you make multiple queries, you learn more about the dataset. And in the limit of infinite queries, you could precisely identify the actual mean of the queries and break the privacy guarantees we've worked so hard to make in the preceding paragraphs. But how can we quantify that?

Let's just start with two queries M_1 and M_2. They have ϵs of ϵ_1 and ϵ_2. So instead of one result, and one probability per neighboring database, we have two, which we need to know the AND of. Because the noise of both queries is independent in each run, the probabilities P(1 and 2) is just P(1)P(2) and thus we get:

$$r = \frac{P\left[M_1\left(D+u\right) = l_1\right] P\left[M_2\left(D+u\right) = l_2\right]}{P\left[M_1\left(D\right) = l_1\right] P\left[M_2\left(D\right) = l_2\right]}$$
$$\leq e^{\epsilon_1} e^{\epsilon_2}$$
$$\leq e^{\epsilon_1 + \epsilon_2}$$

So because we need to multiply all of the relevant probabilities together to get the total probability of observing both l1 and l2, we eventually get a total algorithm where the epsilons just got added together. Which leads to the intuition commonly referred to as something like "adding epsilons". The math here might look very similar to the "Noise and Epsilon Relationship" section above where we looked at a query "vector".

Here's a visual representation:



Two queries on the same database, given to the same person, mean the total epsilon of that situation is the sum of the epsilons of the queries.

Showing this relationship holds for multiple queries looks essentially the same, just with more than 2 probabilities. Showing this works for epsilon-delta privacy is more complicated, and isn't even in the main section of Dwork, instead it's reserved for appendix B; it's quite complicated. Given this is the breezy for-dummies version, see Dwork if you're interested in that more complicated proof.

Privacy Budget: The Final Frontier

Because we can add together the epsilons of multiple queries, this leads to the concept of privacy budget: the maximum amount of information a data curator is willing to release about their data. **People often use privacy budget interchangeably with epsilon, so be cognizant of that**. This leads to a distinction between the query's epsilon which determines how much noise to add to the number, and the total epsilon representing the desired privacy budget.

A simple privacy budget scheme which is pretty common, it seems, is to talk about a daily privacy budget ϵ_{TOTAL} and then each query gets its own individual ϵ_{query}. Which leads to a simple limit on the number of queries that can be performed per day: $N = \epsilon_{TOTAL}/\epsilon_{query}$. Note that you of course also have to think about a budget for delta as well. But in practice, since delta gets defined logarithmically (as in usually your delta is 10^{-4} or 10^{-5}), so additions or small integer multiples don't mean as much and aren't as large of a change.

Is this perfect? No, not at all. Could a sophisticated adversary exploit an even mildly generous daily budget to learn something about an individual over several days? Absolutely! But it's a start and it does well to minimize the damage that a good person could do.

While academia has a lot to say about advanced composition theorems--basically getting tighter bounds on epsilon and delta--how to set privacy budgets **IN PRACTICE** is not, alas, a well studied part of differential privacy in academia.

Different Definitions of "Neighbor"

Implicit in all the calculations above was our definition of neighboring databases: we considered a difference of one user (u) in one database versus the other to be a neighbor of a database. But that's just because it's the easiest way to explain it. Here are some other ways you might define neighbor.

Protect Users

This is what we have been talking about: overall for your ENTIRE database, you care about the user being in or being out: that's your neighboring database definition

Protect Records

You might care more about an event in a database, and less in a user. For example, it might be obvious that a user is in your database (ie that they have a google account or facebook account), but you care more about protecting the records of that user in the database: think about location data points, or maybe ads a user clicked on. In that case, you could define a neighbor as being adding that one record of a user to a database.

You can imagine this would lead to lower sensitivities, and thus less noise needed to be added: the sensitivity of the number of clicks on an ad in the user-neighbor model is infinity unless you bound the contribution of a user, but only 1 in the record-neighbor model.

Protect Partitions

You might also care more about whether a user ends up in a specific group or partition of your database. Say for example, you partition your database by date, and people only really access it by date, individually. Maybe this is app data, and the only sensitive thing is whether a person crossed a bridge on a particular day, and you don't let people access information across days: here you can define a neighbor as being whether a person is or isn't in the database for a given date, and protect your users well enough

Appendices

A Bayesian Aside

Another way of thinking about the probability of getting a certain result for a probabilistic algorithm is in terms of the probability of an outcome GIVEN the ground truth of what database is reality.

$$P[M(D) = l] = P(l|D, M)$$

From here, we can use Bayes' theorem to introduce more information:

$$P(l|D, M)P(D) = P(l)P(D|l, M)$$

So just your standard stats 101 transformation. And actually, the P(D|l) term is what an attacker would be looking for: they want to know what the probability of their target state of the universe is, given what they observed. So let's see if we can isolate that. Now it makes the most sense to take this equation and divide it by the same equation but for a neighboring database:

$$\frac{P(l|D)P(D)}{P(l|D')P(D')} = \frac{P(D|l)P(l)}{P(D'|l)P(l)}$$

$$\frac{P(l|D)}{P(l|D')} = \frac{P(D')}{P(D)}\frac{P(D|l)}{P(D'|l)}$$

$$e^{\epsilon} \geq \frac{P(D|l)}{P(D'|l)}\frac{p(D')}{p(D)} \geq e^{-\epsilon}$$

So assuming the two databases have the same universal "likelihood" of being true [as in P(D)=P(D')], we get the same bound on the Bayesian conjugates as we do on the original probabilities.

Where ln(3) at Google Actually Comes From

It turns out I lied earlier and the coin flip randomized response is not where ln(3) at Google comes from; it's just an interesting coincidence. It actually comes from reasonable-sounding limits on Bayesian certainty increases. Let's take the above equation and move it around a bit for these purposes:

$$e^{\epsilon} \geq \frac{\frac{p(D')}{p(D)}}{\frac{P(D'|l)}{P(D|l)}} \geq e^{-\epsilon}$$

$$e^{\epsilon} \geq \frac{s_o}{s_r} \geq e^{-\epsilon}$$

Where I define the ratio of probabilities as s the “suspicion” initially (0) and after the data release (r). If you simplify this to be a two-possibility world where the user is either in or out of the database and instead we have D_{in} and D_{out}.

$$s_0 = \frac{p(D_{\text{in}})}{1 - p(D_{\text{in}})}$$

$$s_r = \frac{p(D_{\text{in}}|l)}{1 - p(D_{\text{in}}|l)}$$

$$e^{\epsilon} \geq \frac{p(D_{\text{in}})}{1 - p(D_{\text{in}})} \frac{1 - p(D_{\text{in}}|l)}{p(D_{\text{in}}|l)} \geq e^{-\epsilon}$$

$$e^{\epsilon} \geq \frac{\frac{1}{p(D_{\text{in}}|l)} - 1}{\frac{1}{p(D_{\text{in}})} - 1} \geq e^{-\epsilon}$$

$$e^{\epsilon} \geq \frac{p(D_{\text{in}})}{p(D_{\text{in}}|l)} \frac{p(D_{\text{out}}|l)}{p(D_{\text{out}})} \geq e^{-\epsilon}$$

All that to cleanly setup where ln(3) comes from! If the attacker is initially only 50% sure that the the target is in the database $p(D_{in})$=.5, then we want to stop their increase in probability to 75%, $p(D_{in}|l)$=.75. And those numbers giving us ln(3) is left as an exercise to the reader because these kind of fractions works best on paper and not LaTeX: you’re welcome, you get to do some cool fraction math on PAPER now! :D

Additional Reading

Damien Desfontaines has a great series of Blog Posts on privacy techniques