**Thesis Draft**

A significant concern raised by the collection of personal details for research purposes is the storage and protection of these details to ensure that a participant's private information is never disclosed. Additionally, the ability to share research data amongst trusted and relevant parties in a secure way is another requirement that needs to be met to ensure privacy. This introduces a dilemma: how can we share research data in a private manner? The following paper researches this dilemma and proposes to create a piece of software that adheres to the best practices for maintaining participant privacy, whilst also facilitating research data to authorised parties for research purposes.

So the question becomes “how do we analyse data while preserving privacy?”, a question at least 50 years old. A common idea that most people think of is to de-identify the data. That is, to take an original database and move it to a ‘de-identified’ data set. But this brings forth another issue: “de-identified data isn’t”. That is, either the de-identified data is not data, or it is not properly de-identified.

The next idea may be to solely focus on just the statistical side of the data. If all that was required was to perform data analysis, would it not be viable to just release only the statistics of a data set? Suppose that a data analyst were to research the number of people over 180cm tall that like to eat donuts. Is this not a method of research that preserves participant privacy? Due to the fundamental law of information reconstruction, it is not. This is because overly accurate estimates of too many statistics is blatantly non-private, and participants will be identified.

So what kind of privacy guarantee are we trying to make? Is it a breach of privacy if a data analyst is able to learn anything new about patients? This preserves privacy, as the data analyst, after looking at the data set, mustn’t be able to learn anything new about the subjects that they didn’t know before having access to the data set. But if we can’t learn anything new then what is the point of the data set? We’re doing the statistical analysis of data to learn something new about the population.

But is learning something new a privacy compromise? Suppose a participant, subject A, participates in a research experiment. We would have learned the same things had subject A have been replaced by another random member of the population, would we not? This is because we are not learning about subject A, we’re learning about people.

With this approach, we can disentangle learning about the population as a whole from learning about the individuals in the data set. This doesn’t mean that learning about the population as a whole can never hurt an individual; we are not saying that we haven’t learned something that can be used against them.

The English definition of differential privacy says that “the outcome of any analysis is essentially equally likely, independent of whether any individual joins, or refrains from joining, the dataset. This is because we will learn the same things with the same probabilities. Differential privacy separates the harms that can come from the teachings of the database from the harms that can come from participation.

Suppose that a data set teaches us that smoking causes cancer, and that it is known that person A smokes. This person’s insurance premiums may rise; they’ve been harmed by the teachings of the data set. But they would be harmed in this way (the premiums rising) regardless of whether person A is in the data set of out of the data set. Furthermore, learning that smoking causes cancer is the ENTIRE POINT of statistical data analysis. With this in mind, in this scenario, person A can be helped by the data set because they may be inspired to quit smoking.

**Formal definition of differential privacy:**

First we need the concept of adjacent data sets.

Data sets that are almost identical; one is slightly smaller than the other and the larger one contains the data of just one more person.

*M* gives ε-differential privacy if for all pairs of data sets *x, y* differing in the data of one person, and all events *S*

Pr[M(x) [within set symbol] S ] =< e^ ε Pr[M(y) [within set symbol] S ]

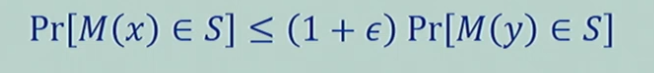
Where:

* M introduces the randomness.
* Algorithms that will analyse data sets will flip coins. They will be randomise algorithms.
* The algorithm/mechanism *M* gives epsilon-differential privacy if for all pairs of data sets *x, y* that differ in the data of just one person (i.e. x and y are adjacent data sets) and every possible output event *S*, the probability that we see S when the data set is x is almost the same as, it is at most RHS of the equation: it is at most e^epsilon \* the probability that we see S when the data set is *y.* so the ratio of these two probabilities is at most e^epsilon. And the probability space is over the coin flips of the algorithm.

The algorithm is the good guy.

And because we said for all pairs of the data sets, the same inequality would hold if we were to swap x and y.

* + This definition was put forwards by Dwork, McSherry, Nissim and Smith in 2006.
* So e^epsilon when epsilon is small is about (1 + epsilon):



* So when epsilon is 0, it says that these 2 probabilities must be the same; when there is 0 probability loss, we also get 0 information about the data set.
* So, a subject may say “if a bad event is very unlikely when I’m not in dataset (y), then it is still very unlikely when I am in the data set(x).
* So epsilon is our measure of “privacy loss”.

Properties of differentially private algorithms:

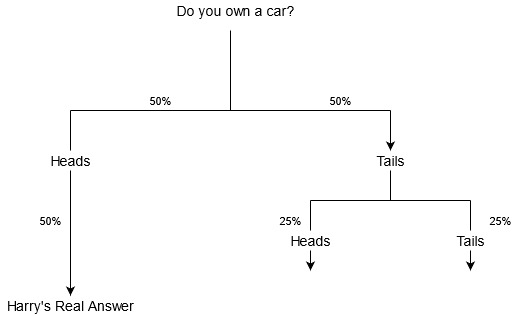
* Future-proof:
  + Privacy comes from the process by which the outputs are generated, and if a data release is differentially private, that ratio of probabilities doesn’t change if someone who sees the output learns other auxiliary information or sees other data sets that will exist in the future.
* Automatically yields group privacy:
  + K\*epsilon for groups of size k; .
  + If an algorithm yield epsilon-differential privacy for a single individual, then it automatically yields k\*epsilon differential privacy for groups of size k. so if epsilon-dp is satisfactory for one person, it will be scalable for k people as well.
* (most importantly, Dwork says) Understandable behaviour under composition:
  + Understand the behaviour of privacy loss of DP under composition.
    - Can bound cumulative privacy loss over multiple analyses.
  + “the epsilons add up”.
  + “one crude bound shows us that in the WCS the epsilons add up; we can do better than that and that will be the subject of Guy Rothwells talk.”
* Programmable:
  + Because of the composition property which says that we can understand when we ask about many different statistics, we get that differentially private algorithms are actually programmable. That is to say in ordinary programming, you don’t just compute everything from ands and ors. What we do is build sub-routines and algs for smaller problems and the whole field of alg design is the creative and intelligent combination of different computational building blocks to carry out a sophisticated computational task while minimising some sort of resource usage (time or space).
  + The same thing is true w DP. You can take small differentially private building blocks and combine them to carry out complex Differentially P data analyses. And that is a power that no other approach has [as far as Dwork knows].

Differential privacy has to somehow blur the answers a little bit. So we have truth and the blurred version of truth.

Let’s now look at an example of differential privacy in use. Say we have conducted a survey to determine the amount of people that own a car. Suppose that the survey has had participants select one of two options: ‘yes’ or ‘no’. Suppose also that all of the collected participant data has been stored on a server.

However, suppose that instead of sending the server the real answers, we have introduced some noise. Let’s say that Harry owns a car and has selected ‘yes’. Before his response is sent to the server, our differential privacy algorithm will randomise Harry’s answer by using coin flipping. If the coin returns ‘heads’, then Harry’s real answer is returned to the server. If the coin returns as ‘tails’, however, then the algorithm will flip a second coin and send a ‘yes’ to the server if the second coin returns as ‘heads’, and a ‘no’ to the server if the second coin returns as ‘tails’.

The server will see the data coming in, but because of the added noise individual records are untrustworthy. Our record for Harry may say that he owns a car, but there is at least a 1 in 4 chance that he does not own a car, yet the algorithm has returned that he does. Now, because we know how the noise is distributed, we can compensate for it and end up with a fairly accurate view on the amount of people that own cars. However, this is just a simple randomisation algorithm. Many real-world algorithms utilise the Laplace distribution to spread data over a larger range and increase the level of anonymity.



* Modelling method used in structured systems analysis to create analysis models and, very often, the requirements model during the system specification stage. These methods centre around modelling both the system and subject worlds, although they mainly use the system world terms. Consequently, they require analysts to have particular skills in expressing subject world requirements in system world terms.
* Models in structured systems analysis are made up of three components:
  + The process
  + The data
  + The system functions
* Data flow diagrams model system processes and are one of the most important modelling tools used by systems analysts.
* DeMarco (1978) and Gane and Sarson (1979) suggested that data flow diagrams should be the first tools used by systems analysts to model system components.
  + Where these components are the system processes, the data used by these processes, any external entities that interact with the system, and the information flows in the system.

**DFD Symbols**

* Symbols are used to represent systems.
* Most DFD’s use four kinds of symbols to represent four kinds of system components:
  + Processes
  + Data stores
  + Data flows
  + External entities

**Processes**

* Processes show what systems do.
* Each process has one or more data inputs and produces one or more data outputs.
* Processes are represented by circles in a DFD.
* Have a unique name and number which appear inside the circle of the process.

**Files or data stores**

* A file or data store is a repository of data.
* Processes can enter data into a data store or retrieve data from the data store.
* Each data store is represented by a thin line in the DFD and has a unique name.

**External Entities**

* External entities are outside the system, but they neither supply input data into the system nor use the system output.
* These are the entities that the designer has no control over.
  + E.g. an organisation’s customers or other bodies with which the system interacts.
* Alternatively, if we are modelling one section in an organisation, other sections are modelled as external entities.
* External entities are represented by a square or rectangle.

**Data Flows**

* Data flows model the passage of data in the system.
* They are represented by lines joining the system components.
* The direction of flow is indicated by an arrow, and the line is labelled with the name of the data flow.
* Flows of data in the system can take place:
  + Between two processes.
  + From a data store to a process.
  + From a process to a data store.
  + From a source to a process.
  + From a process to a sink.
* we have no control over flows between external entities so we do not model them.