Differentially Private Data Querying

Honours Project
BSc. (Hons.) Applied Computing

Sam Edward Angus Hood
070017432

Supervised by Dr. Marco Gaboardi

2014

Differential privacy is a way of protecting the identity of data cohorts whilst providing useful information to interested parties, by applying noise to aggregate queries. There are two main aims for this project; the first of which is general research into differential privacy, to determine whether it has a viable place in the context of smart-homes, and at what point the data becomes too noisy to be usable. The second aim is to create a website that would function as a place to experiment with differential privacy; allowing the user to perform aggregations on both simple and home data and compare the results to non-differentially private querying.

Differentially Private Data Querying

Samuel E. A. Hood
Honours Project
BSc. (Hons.) Applied Computing
University of Dundee 2014
Supervisor: Dr. Marco Gaboardi

Abstract *– Differential privacy is a way of protecting the identity of data cohorts whilst providing useful information to interested parties, by applying noise to aggregate queries. There are two main aims for this project; the first of which is general research into differential privacy, to determine whether it has a viable place in the context of smart-homes, and at what point the data becomes too noisy to be usable. The second aim is to create a website that would function as a place to experiment with differential privacy; allowing the user to perform aggregations on both simple and home data and compare the results to non-differentially private querying.*

1. Introduction

Our modern lives are defined by data; every minute more than 2 million Google searches are performed, more than 1.8 million things are liked on Facebook, and 72+ hours of video is uploaded to YouTube (Qmee, 2013). The presence of computers in our daily lives has led to the storage and utilisation of extensive and complex data covering anything from medical and police records, to second-by-second statistics of home temperature and power usage.

Data is everywhere, and with our planet moving towards an "Internet of Things" (i.e. objects connecting to the internet, such as light bulbs or flower pots), the future will bring with it huge amounts more. The amount of data that is continuously being generated provides us with an almost real-time data representation of our world, allowing us to react to situations and analyse our planet with increasing efficiency, accuracy, and efficiency.

1.1. Smart-homes

One outcome of 'things' being able to connect to the Internet and upload data, is smart-homes; houses that are outfitted with dozens of sensors, integrated into home systems, which provide a data representation of the current state of many aspects of the home, such as temperature, power usage and a light levels. This wealth of data allows a broad range of analyses which has benefits of allowing unprecedented control of the home – inhabitants can, for example, have their home learn their living routines and adjust the environment accordingly, depending on various factors – a good example of this is a central heating system that adjusts the home temperature depending on if there is anyone at home (inferred from perhaps movement sensors, light levels, or heuristics). Control of a home in a 'smart' way has two main advantages for the inhabitant, the first of which is allowing their home to be more eco-friendly, for example a reduction in central heating activity to only be activated when it's needed, and not being accidentally left on all day while the inhabitant goes to work. Another main advantage is automation, meaning that the house is much easier to manage; an example of this is having a fridge/cupboard that automatically keeps track of the inventory and generates shopping lists on the fly.

Utility companies also have a great interest in smart-homes, allowing them to accurately bill the consumer without having to be in constant contact with them, or send around staff for meter-readings. The companies can also collect statistics on their customers with great ease, allowing them to intelligently distribute their services to areas that need it most, with a high temporal resolution.

1.2. Privacy Concerns

While this constant and ever-growing stream of data is incredibly useful to those who seek to use it for the previously mentioned feats, it also has a big downside that comes in the form of privacy. Datasets can be used for malicious purposes if they are not made sufficiently private; access to the data can reveal huge amounts of information about the individual's daily habits and routines. An example of this would be attackers determining statistically the best time to break into a home based on data gathered about an individual – what hours they work, if they often take holidays (if so, for how long), and various other metrics that allow attackers to build up a profile of someone and find the weak spots.

Inherently, privacy is a key priority in our "Internet of Things". The most obvious way to make a set of data private is to essentially erase it – if there's no data left, attackers have nothing to work from. This isn't particularly useful however, as the data then becomes useless to parties who have no malicious intent. Evidently there has arisen a trade-off of utility for privacy, with privacy increasing as utility decreases. A common way of providing privacy to data is reducing the amount of information that can be ascertained through queries – statistical databases intend to provide maximal utility, whilst protecting the identities of

individuals in the data, normally by allowing only aggregate methods to be performed, and not allowing direct access to the database behind. However these databases are still open to attacks through various means.

In this report we cover what form these attacks take, and investigate the effectiveness of differential privacy in evading these attacks, both in a general setting (the theory of differential privacy), and in the context of smart-homes. We also investigate the PINQ platform, developed at Microsoft Research as a viable platform for providing differential privacy, and cover the process of integrating it with a website that provides tools for performing experiments with differential privacy.

2. Background

In this section we will firstly cover k-Anonymity, which is a method designed to preserve privacy by removing personally identifiable information from data. However, k-Anonymity does not protect the identities of the data cohorts when the attacker has additional knowledge (as shown in the "Netflix Prize" case). Hence we introduce the role of differential privacy in avoiding privacy breaches, when the attacker has additional information, and cover how the mechanism works.

2.1. K-Anonymity

In order to protect the data cohort (that is, the identities of the individuals associated with certain data), a logical step is to suppress or generalise personally-identifiable information in said data – replacing column values such as name or race with values such as an asterisk, and generalising other columns (e.g. a value of "25" could be generalised as "between 20 and 30"). K-Anonymity does exactly this, and aims to increase the privacy of the data cohorts, whilst providing useful data on which to do analyses (Sweeney, 2002, pp. 1-3).

Protecting data with k-Anonymity doesn't necessarily mean that the data is free from the possibility of attacks; additional information released separately to the dataset can be compared against the anonymised set, hence re-identifying more about the individual than was intended. (Sweeney, 2002, pp. 10-12)

2.2. Netflix Prize

The Netflix Prize is a good example of why consideration of additional information is important in protecting the privacy of data, despite the data having all personally identifiable information removed. In an effort to increase the effectiveness of their recommendation algorithm (used to determine which movies or TV shows a user would be interested in based on previous views), Netflix started a competition in 2006, urging teams to try and improve their algorithm. There were some security and privacy issues over this; in 2007, researchers from the University of Texas were able to compare the data that Netflix had provided with the Internet Movie Database (IMDB) – the 'additional information', successfully being able to identify several individuals in the process by comparing similar ratings for films. This led to four Netflix users filing a lawsuit in 2009, and the end of the competition.

2.3. Differential Privacy

K-Anonymity can only take you so far on the road to privacy; suppressing and generalising columns in your dataset will only protect the identities of those involved to a degree; it does not take into consideration the additional information that can be used to identify data cohorts. To illustrate how one of these attacks would work, let's take a list of numbers from 1 to 10 (D_1):

$$\mathbf{D_1} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Taking the normal average of D_1 requires taking the sum of the list and dividing by the number of values; the average of D_1 would be **5.5**. Now, suppose we remove the **2** from the list to produce a new data set (D_2):

$$\mathbf{D_2} = \{1, 3, 4, 5, 6, 7, 8, 9, 10\}$$

As you would imagine, taking the average of D_2 will result in the slightly higher value of ~**5.89**. This reveals information about the data behind the scenes, particularly when someone has additional information; learning the average when a record is removed, one could imagine how simple it could be simple to deduce the exact values of the removed record, and consequently, all of the values in D_1, despite never having had direct access to it.

To put this example in the context of personal data, we can take an imaginary medical data table (see Table 1 below) that contains a list of people's names, and their systolic blood pressure:

Name	Systolic BP
Jim	119
Sue	145
Dave	90
Eric	92
Nancy	149

Table 1: Individuals and their Systolic BP

Let's say that we have no direct access to this data, but we are able to perform aggregate functions on it; if we take the average systolic BP for the entire cohort, we get **119**. If we remove **Dave** from the data, we get an average of **126.25**; from this we can infer that **Dave** has a *low* systolic blood pressure compared to the average of the original cohort (the removal of his BP increased the average) – since the original average BP (**119**) is incidentally a normal systolic blood pressure, we can

make the assumption that **Dave** likely has hypotension, without ever seeing directly the BP associated with him.

In the real world, the attacker wouldn't be able to see the names associated to the records within the dataset, as they will be hidden by methods such as k-Anonymity. This example can only work when the attacker has external, additional information – a good example of this would be that the attacker knows the initial average blood pressure of the data, and then discovers that **Dave** just requested the removal of his records from the dataset. Upon inspection of the data again, with the knowledge that **Dave** has left, they will be able to infer facts about him.

It's fair to say that this kind of inference is not desirable in a set of data that is supposed to not reveal anything about the individuals involved. One way to prevent attackers from learning this kind of information is to add noise to the data (generally, Laplace noise is used due to the inequality required by differential privacy), meaning that results provided by the data will not be exactly correct. This means that, referring back to Table 1, the average of the list has a *probability* of being around **126.25**, based on the amount of noise added. Similarly, the average of the dataset with one record removed has a *probability* of being around **119**, again based on the amount of noise added. This has an effect on both the accuracy and privacy of the data by covering up true values and only providing a probability of a correct result, of which is controlled by the noise. It's important to also note that the more data being used, the more effectively differential privacy is at preserving the privacy of the data – this is because the removal of one record from a huge dataset will already have a small impact on the overall average, thus making it easier to cover up with noise.

Imagine we have a differentially private average function, which adds Laplace noise on a set of data, with the peak of the distribution being the true average (**126.25**). In the context of the blood pressure example, we have the first query on the full dataset; our differentially private average query provides us with the result: **124**, found just below the peak of the distribution (see Figure 1 in blue).

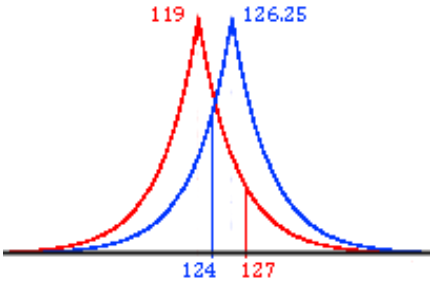

Figure 1: Laplace distributions around the true average on unaltered (Blue) and altered (Red, record removed) dataset

Now that we have the 'noisy average' of the full dataset, we can remove a record as before, and run the query again, this time the distribution is centred on another average (**119**) (see Figure 1 in red). The differentially private function returns a value above the peak (by chance): **127** – interestingly, the average of the altered dataset appears greater than the average of the unaltered dataset, despite the fact that the actual average for the altered dataset is lower than the unaltered dataset. This happens purely from the fact that there's now a probability that lower or higher results from the actual average will be calculated, since noise has been added. Here is where the privacy is preserved; previously we investigated the dataset using true averages and were able to discern information that was not there by removing records (the fact that Dave likely has hypotension). Now, using differential privacy, we can't tell for sure if Dave has hypotension, as repeated queries may be lower or higher than the noisy average of the unaltered dataset – meaning no true information can be inferred simply from comparison of these two datasets using a single query.

A query or result is said to be differentially private if *the removal (or addition) of one record has limited impact on the overall result*. This definition is formalised by Cynthia Dwork as follows:

Definition (Dwork, 2008, p. 2) – A random function K (which adds noise to the data) is (ε, δ)-*differentially private* if for every two data sets (D_1, D_2) differing by one element, and for every possible observation $S \subseteq Range\ (K)$:

$$e^{-\varepsilon} \leq \frac{\boldsymbol{Pr}[\boldsymbol{K}(\boldsymbol{D_1}) \in \boldsymbol{S}]}{\mathbf{Pr}[\boldsymbol{K}(\boldsymbol{D_2}) \in \boldsymbol{S}] + \boldsymbol{\delta}} \leq e^{\varepsilon}$$

This implies for any possible observation that the ratio of the probabilities given by K on D_1 and D_2 is bound by $\mathbf{e}^{\varepsilon}$.

Epsilon controls how much noise is added to the dataset, the lower epsilon is, the more noise is added.

2.4. PINQ

Privacy Integrated Queries, or PINQ for short, is an extensible platform that guarantees unconditional differential privacy on data (McSherry, 2009). PINQ uses SQL-like queries, through an extension of Microsoft's LINQ (Language Integrated Query). PINQ provides a platform to query data using several 'noisy' aggregations (average, median, and sum), which provide differentially private results by adding Laplace noise to the data.

PINQ works by firstly filling an `IQueryable<T>` object with the source data, then an aggregation method can be called to perform queries on the data. For example, the `NoisyAverage()` method takes the following parameters:

```
double epsilon,
Expression<Func<T, double>> function
```

Epsilon determines how much noise is added to the results of the query, and Function is a lambda function to be applied to every value in the source data; normally this is a conversion to a double. Therefore a method call to get a differentially private "noisy" average with an epsilon of '1', provided the source data has been previously provided, would be:

```
PINQ.NoisyAverage(1, x => (double)x);
```

Once the lambda function has been applied to the source data, the values are normalised between -1 and 1; in the current release of PINQ there is limited functionality for queries – in particular it's restricted to data that is between -1 and 1. It was agreed that this would need to be extended to accommodate data of all sizes.

PINQ then performs the actual averaging; it calculates the sum of the normalised values and adds Laplace noise by providing epsilon to a Laplace method, this is then divided by the number of values. If the resulting value is outside the normalised bound of -1 and 1, the averaging is calculated again – this happens until the result is within the range of -1 and 1. The result is then scaled back up to the original range and returned to the parent method.

2.5. Smart-homes

Using differential privacy to protect home data means that one can never get *truly* accurate value of, say the temperature within the house; close enough that it's useful, but noisy enough that you cannot use it along with additional information to identify more than simply the temperature. Of course, given enough single queries, you will be able to learn the noise and infer this.

Understandably, privacy is a number one concern when it comes to handling the data generated by a smart-house. The data can reveal surprisingly accurate information about the day-to-day activities in the house; this was investigated by a team at the University of Massachusetts Amherst, where they showed the minute detail that could be discovered from patterns within the data, whether it be if there's a new-born baby in the house (more power usage in the middle of the night – parents getting up to feed the baby), to whether or not the owners had a hot breakfast that morning (more power usage in the morning) (Greveler, et al., 20XX).

Weather data can also be collected by homes, which can act as pseudo-weather stations, to help fill in weather data that aren't covered by real weather stations. However, weather data collected by houses can be cross-compared against the additional information of national weather data to deduce geographically where the house is – the more weather information that is created (humidity, wind speed/direction, etc.), the easier it is to pinpoint.

3. Problem Identification

3.1. Current state of Differential Privacy

Having been around for about 5 years, differential privacy is still in its infancy, information about it is relatively sparse; limited to a handful of research papers, and of course PINQ's implementation. Differential privacy in the context of smart houses has been investigated, but it still remains relatively difficult to experiment with differential privacy without building an ad-hoc solution.

3.2. Aim

There are two main aims for this project; the first of which is general research into differential privacy, to determine whether it has a viable place in the context of smart-homes. The second aim is to create a website that would function as a place to experiment with differential privacy; allowing the user to perform aggregations on both simple and home data.

*3.3. Doll's House and Smart**

Initially it was planned to use the Doll's House at the School of Computing, Dundee University, to provide the data, since it would be useful to use the School's own projects, allowing us full control of the home data. Previously another student had developed software that generates realistic Doll's House data without having to use real people – the idea is that you provide routines, which are made of actions, to define days of the week, these then have slight noise added to them and generate any number of weeks of data (Walker, 2013). It was intended that we would use this software to generate several million rows of data to use, but unfortunately, the amount of data it would generate was low compared to what we expected (400,000 records for a year as opposed to our hope of more than 4,000,000). By this point a large chunk of time had been dedicated to inputting the individual routines, which was proving to be a lot of work – the consensus was to scrap the Doll's House idea and find a pre-existing dataset, as time constraints were closing in.

We investigated the paper which discovered patterns in smart home data (Molina-Markham, et al., 2010), and found the dataset which had been used – luckily this dataset was for research and open to the public. Smart* is a project that aims to optimise home energy usage, and for this, the team at the University of Massachusetts Amherst have designed an actual live smart house that actively collects data, and have uploaded the data for free use (UMass, n.d.) (Barker, et al., 2012).

The Smart* home provides a huge amount of data for various sensors around the house, the sensor types are as follows:

- Circuit
- Door
- Environmental
- Furnace
- Meter
- Motion
- Phase
- Switch

Altogether the Smart* dataset provides a total of just over 32 million records, which is more than sufficient for our needs.

3.4. Typical Users

Users of the site would already have some understanding of differential privacy, and would use it to see how it works when applied to real data. It would be expected that the typical users would be researchers and students.

3.5. Site requirements

The site is in two main parts; simple data querying, and home data querying, each having their own requirements. The simple data provides a way of performing differentially private aggregations on simple data lists, that is, lists of numbers in a range (in this case we restricted the usable values to the range **0 - 100**). Home data querying provides similar functionality, but uses actual home data instead of lists of simple numbers.

Generally the requirements of the site are fairly informal; this is because there is no 'client' that is having the site made, so mostly the requirements were created as and when we needed them. For example, in the list below, delta (Functional **1.6** & **2.7**) was added near the end when it was decided that it would be interesting to add the option to use Gaussian noise as well as Laplace; both of which are standard in obtaining differential privacy, Laplace using epsilon, and Gaussian additionally using delta. While the requirements are informally created, they are designed to fulfil the needs of a typical user.

The decision to restrict the home data to temperature, humidity, power and rain rate was that of simplicity; it wasn't required that the website analyse every possible metric of the smart home data, but only a handful in order to demonstrate differential privacy.

Functional:

1. Must allow the user to input simple data queries:
   - **1.1.** Data (list of numbers)
   - **1.2.** Epsilon
   - **1.3.** Iterations
   - **1.4.** Query Type (average/median)
   - **1.5.** Noise Type (used in PINQ; Laplace or Gaussian)
   - **1.6.** Delta (used for Gaussian noise)
2. Must allow the user to input home data queries:
   - **2.1.** Data (temperature/humidity/power/rain rate)
   - **2.2.** Timespan (month/week/day/hour)
   - **2.3.** Epsilon
   - **2.4.** Iterations
   - **2.5.** Query Type (average/median)
   - **2.6.** Noise Type (used in PINQ; Laplace or Gaussian)
   - **2.7.** Delta (used for Gaussian noise)
3. Must generate charts based on input, providing differentially private results using PINQ
4. Shall calculate non-private aggregations for each query and display them alongside charts (e.g. actual average displayed next to differentially private results)
5. Shall provide the user with instructions on how to use the charting
6. Shall provide background information on differential privacy including links to papers

Non-functional:

1. Must be easy to use
2. Must work on all main browsers (Chrome, Firefox, IE, Safari)

4. Design

4.1. Methodologies

Due to the research-oriented nature of the project, a typical 'user-centred' approach was not used; instead something more like RAD (Rapid Application Development) was used to iteratively produce parts of the site depending on our needs as they arose. If something was taking too long to produce or was deemed unnecessary, it would be scrapped and the next part would be worked on.

Project management was provided by Trello, which works like an agile board. Main pieces of work were added as tasks, and subtasks were added as required. The rationale behind using this as opposed to classic project management tools such as a Gantt chart, is that there would be too many changes made to merit using a single Gantt chart; using Trello meant that when there was an idea coming from experimentation, it could be added as a task (or subtask) on Trello, allowing a more fluid, but controlled, project management. Time constraints were placed on the main tasks (e.g. develop website by 20th April), and no time constraints were added to secondary tasks (e.g. optimisation, which can only happen if the site was finished in time).

4.2. Pre-design Experimentation

Before making decisions on how the site would be implemented, time was taken to investigate the PINQ library and find out how it works – this was done as a Windows Forms application using C# since it was the suggested language for the library (PINQ is heavily reliant on LINQ, which is implemented in both C# and Visual Basic – C# was the preferred choice of these, mostly due to experience using it regularly).

A couple of small pieces of software were produced which were used to find out if PINQ is a viable platform to implement differential privacy in our website – the experimentation led to several design ideas and a few discoveries about PINQ that would have drastically increased the development time of the website had they not been found prior.

The first of these pieces of software is a very simple application that intended to investigate the results that PINQ provides for basic data sets. This software is purely experimental and has not much structure, it was used as a sandbox to experiment with PINQ – it is included in the project files but is not well commented or structured.

4.3. Clustering

It was realised that there would need to be two different types of home data query:

- Simple Query
- Clustered Query

The simple query would be for questions such as "what is the average power used per week?" which would require almost exactly the same computation as the simple data (basic lists of numbers); take all of the power readings during a week as a list, and perform a noisy average on it several times. The output of this would be a chart that shows a distribution of values around the actual average power.

Clustered queries are a little more complex, and are for questions like "what is the average power usage per minute for an entire week?". For this, the output would be a chart that shows the average power for one minute as a bin, and the chart would cover a week's worth of data. The rationale behind needing these clustered queries is to simulate the results found in the paper on smart metering from Massachusetts (Molina-Markham, et al., 2010).

A clustering algorithm was developed that would take a list of data containing timestamps, and turn it into a list of lists, each outer list being a cluster, and the inner lists being the values in that cluster.

```
var cluster = new List<T>();

foreach (var item in data)
{
	if (cluster.Count > 0 &&
		item.Timestamp >
		cluster[0].Timestamp
		+ timestampDelta)
	{
		yield return cluster;
		cluster = new List<T>();
	}
	cluster.Add(item);
}
```

In order to provide a quick analysis on all iterations, it was decided that the factory design pattern should be used to spawn a thread for each iteration. This factory would be the `PINQAnalyser` class, which repeatedly spawns threads that perform analysis using PINQ.

4.4. PINQ Issues

During investigation of the PINQ platform, we were surprised to see that it wasn't working entirely correctly, and was producing random results between -1 and 1 (see Figure 2 below) that didn't line up to anything in the original data.

The experimental software was used to produce these results; evidently this was an important step considering that if this was found at a later point, there would have been many issues associated with it.

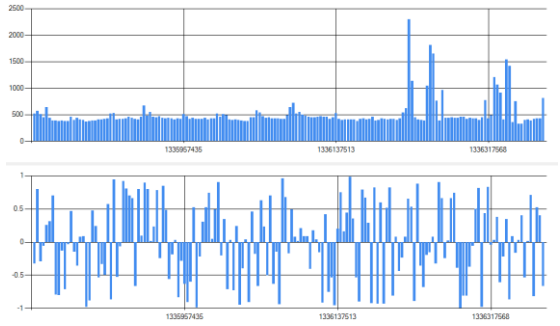

Figure 2: PINQ random noise issue

Figure 2 is the result of a clustered query for average power usage per hour for a week; the top chart is a normal average performed on each cluster, with no privacy involved, and the bottom chart is the result of a differentially private "noisy" average on each cluster. As shown, the values in the bottom chart are all over the place, and don't seem to represent anything from the top chart. There were two ideas as to why this was the case, either the privacy settings were being used wrongly and in fact we were getting back data so private that it was just noise, or there was an issue with PINQ itself. On investigation of the source code of PINQ, we found the issue, which is in regards to the normalisation of values between -1 and 1 which happens before an average is calculated. This was implemented using the following code:

```
IQueryable<double> values =
source.Select(function)
.Select(x => x > +1.0 ? +1.0 : x)
.Select(x => x < -1.0 ? -1.0 : x);
```

This code uses three lambda expressions to alter the data provided in the source; firstly, the 'function' is the user-defined function that's passed into the method (usually conversion to double), the second two alter each element such that if `x` is greater than 1, the value is set to 1 – similarly, if `x` is less than -1, the value is set to -1.

As shown (see Table 2 below), this resulted in an almost complete loss of data in the situation that the data values regularly exceeded the range of -1 to 1. Included is the expected output that we imagined would be produced, provided that the normalisation scales the minimum and maximum values to -1 and 1 respectively:

Input	Expected	Output
1	-1	1
2	-0.77...	1
3	-0.55...	1
4	-0.33...	1
5	-0.11...	1
6	0.11...	1
7	0.33...	1
8	0.55...	1
9	0.77...	1
10	1	1

Table 2: Expected and actual PINQ normalisations

We came to the conclusion PINQ was developed to analyse data sets whose available data fell between -1 and 1. Considering that if values fall between -1 and 1, the value would be left alone, one can imagine that the original data set was a set of very small values, and the 'normalisation' was intended to essentially trim the values down.

As soon as any data that isn't between -1 and 1 is used, this breaks down, and you have your dataset reduced to a set of single values – removing almost all information that the data contained.

It was agreed that this needed to be changed to accommodate our data; the code was amended to perform a basic linear transformation on the values, scaling the set down using the minimum and maximum values to re-compute all to between -1 and 1 when normalised:

```
var upperValue = source.Max(function);

var lowerValue = source.Min(function);

IQueryable<double> values =
source.Select(function)
.Select(x => (x - lowerValue) /
(upperValue - lowerValue) *
(1.0 - (-1.0)) + -1.0)
.AsQueryable();
```

As hoped for, this code resulted in the expected values shown in Table 2. It was later on agreed that using the maximum/minimum values meant that those were being shown on the chart output, which is revealing information about the data that should be private (the data behind the scenes). Consequently this was changed to be clamped to static values of 0 and 100, removing any possibility that values can be inferred from the values shown on the output chart.

Additionally, to fix the issue of returning values between -1 and 1, PINQ was further changed to restore the calculated value to the scale of 0 – 100 before returning a result.

4.5. Tools/software

The pre-design software confirmed that C# would be the right tool for the job due to its simple and integrated LINQ implementation. This in turn indicated that ASP.NET would be likely the best platform to use for the website, since a lot of the code could be reused or translated easily. Originally it was considered that ASP.NET Web Forms would be used as it was the comfortable choice, however there was interest in utilising the relatively new MVC4 platform (MVC5 has recently been released, but would be unsupported by the server that was chosen to run the application), considering that there would be emphasis on moving fairly large amounts of data back and forth between the client and server – MVC allows you to represent data models that are output and amended on a view, which is perfect for keeping track of several charts in one object, and simply passing it back and forth. The site was developed locally on a laptop, at various stages the project was published to the webserver, and changes committed to a Github repository. A big aspect of choosing a version of MVC was the usage of the Razor view engine, which is a server-side mark-up language similar to the ASP.NET mark-up language, with the exception that it allows validation to work automatically between the client and server.

For example, the model may have some attributes such as "Name", using the data annotations library, we can set this attribute to have a certain validation associated with it. On the view, it's simply a case of adding the following line of code at the point that you require an error message to pop up (which is also specified in the model):

```
@Html.ValidationFor(m => m.Name);
```

This allows the developer to easily change error messages and validation methods without having to get involved with the view whatsoever – meaning that this can be changed on the fly without bringing the site down. While this is not something that will be happening with this project, it provides the benefit of allowing the views to be much more readable, without the mess of validation messages and such.

For most visual effects, and general JavaScript work, jQuery was used – a massive JavaScript library that provides simpler syntax and abstraction away from pure JavaScript. Notable uses of this in the project is the popup boxes for logout, add charts, and loading; utilising jQuery's `.hide()` and `.show()` methods. jQuery is also used to provide AJAX functionality to

MVC – allowing use of the Ajax.BeginForm method within MVC, providing simplified AJAX requests via form input.

The charting used by the site was provided by Highcharts, a JavaScript library that provides a huge number of charting functionalities (Highcharts, 2014). Originally it was planned that we would use the C# charting library, but this seemed incompatible when used on the web – Highcharts was a much more durable alternative. The charts exist within the server back-end, where they are generated and populated, and are sent back to the view in the form of a model, which is then used to display on the page using Razor syntax.

Nuget is a package manager that is designed to make it easy to keep all references up to date within the project – while this may not be necessary entirely for a short project, it's good practice to keep it manageable for future builds. The main packages handled by Nuget is the various jQuery libraries used by the project, Highcharts is not managed by Nuget as changes to the framework may break the charting so a static and unchanging JS file was used instead.

Class and sequence diagrams were produced using StarUML, freeware software that provides all common UML diagramming functions.

Additional software was used over the course of the project to fulfil small tasks; Photoshop CS3 was used to produce UI elements such as the logo and favicon used on the site, Notepad++ was used for general note-taking and small amendments to server files that wouldn't justify a full re-publish.

4.6. Home Queries and the Database

SQL stored procedures were required to interface the website and the Smart* database. A single stored procedure was designed so that data could be pulled for temperature, humidity, or rain rate, and for a specific duration (hour/day/week/month). No aggregations or transformations were allowed to be performed on the data as it was pulled, in order to restrict all computations on the data to PINQ alone – ensuring that the SQL had no effect on the results. The output of the stored procedure is a list of doubles that represent whatever was requested.

The rationale behind creating a single query instead of several is to reduce the amount of work the web application needs to do to retrieve data – considering that the data is not as important as the analysis. The query takes two parameters: `@category`, which can be one of either 'temp' (temperature), 'windspeed', or 'humidity', and `@timespan`, which is the amount of second's worth of data that is to be retrieved. The timespan is restricted by the web application to hour/day/week/month (3600 / 86400 / 604800 / 2629740 seconds, respectively). Table 3 outlines the statistics of each combination of query.

Category	Timespan	Records	Time (s)
Temperature Wind Speed Humidity	Hour	11	0
	Day	287	0
	Week	2,015	0
	Month	8,754	0
Power	Hour	7,286	6
	Day	214,593	6
	Week	1,687,162	25
	Month	7,879,564	43

Table 3: Record counts and time taken for data retrieval

Temperature, wind speed, and humidity are all from the 'Environmental' table, meaning that they all have the same number of records returned. Power is from the 'Circuit' table, and has a massive amount of records compared to the others; the problem with this is the retrieval times, any request for a week or above of data begins to take a very long time – this will have an impact on the performance of the website.

Below is a sample of the SQL stored procedure covered previously; the excerpt is the wind speed data retrieval. Note the `@minTimestamp` variable, this is used to determine where to start the chunk of data at – for the purposes of this project it was set to the first timestamp in the table.

```
IF @category = 'windspeed'
BEGIN
	SET @minTimestamp =
	(
		SELECT MIN(TimestampUTC)
		FROM Environmental
	)

	SELECT windSpeed 'output'
	FROM Environmental
	WHERE TimestampUTC
	> @minTimestamp
	AND TimestampUTC
	< @minTimestamp + @timespan
END
```

4.7. Hosting

The site is hosted on Arlia, a server at the School of Computing, University of Dundee, which supports ASP.NET MVC4 up to .NET version 4. Direct access is available using 'arlia.computing.dundee.ac.uk/2013-projects/samhood'.

The domain name 'differentiallyprivate.com' was registered using the domain site 123-Reg.com. CNAME records were added to the DNS listings of the domain to point to the Arlia server. The rationale behind deciding to buy a domain name for the project was that of professionalism, the site looks much better

with its own domain name than the server URL – and also allows potential development in the future.

The database is located on the Namek server; this requires VPN access to use, but luckily Arlia is on the same network so extra configuration wasn't needed to allow communication between the two.

5. Implementation

5.1. From Design to Implementation

The implementation of this project was experimental by nature; there was only so much we could design before implementing it and seeing if it would work.

The majority of the work in this project is centred around the charting mechanisms for the website; a typical rotation of design/implementation would involve firstly thinking about what we need, initially this was the ability to investigate PINQ, which was then implemented and used to discover what was next required. These stepping stones allowed a lot of control of the project as it progressed, however the downside of this method was that there were potential dead-ends to be encountered.

In addition to the website, small pieces of software were developed to investigate PINQ and solve issues – the C# Windows Forms application which was created to experiment with how PINQ works, the logic of which was ported to the website when it had been worked out. Additionally the CSV fixer was produced to solve the issue with missing values in the Smart* data.

5.2. Architecture

The main body of the site is in the charting functionality – the site was designed to have a generic PINQ analysis interface, which works for both the home data and simple data. This section will cover the architecture as a use case of the charting functionality more than the entire site; this is because the rest of the site is simple content as opposed to the complex nature of the charting. References to 'chart object' means an implementation of the class that is responsible for storing query parameters, and holds the actual chart.

All server requests in the context of charts are handled by the chart controller; in response to a GET request with no parameters; this involves setting up an empty list of chart objects (referred to from here as '`MultiChart`'). One chart object is added with default parameters and the entire `MultiChart` is packaged into a view and sent to the client. This functionality is also used when the user clears all queries.

It's important to note that the client's page will take the last element of the `MultiChart` as being the one that is currently being 'added', the ones that come before that are dealt with as being already added to the list.

A typical query would start by the user entering the parameters to be used (functional requirements 1 & 2), which may vary from the served defaults. This data is then added to the client-side `MultiChart` and is posted back to the server to verify the data and add it to the server-side `MultiChart`.

If the server detects that it's a POST request, this indicates that the user has submitted a new query to add to the list. The server will verify the query parameters are within their respective ranges, if anything is wrong, the model is sent back to the client with an error. If the parameters are verified, the server simply creates a new chart object, adds it to the `MultiChart` and sends it, inside a view, back to the client.

Once the user is happy with their chart parameters, they can start the calculation. The decision was made to make this part entirely AJAX powered, in order to stop the browser from sitting and loading for the entire duration. If the server detects an AJAX request, it sees it as a trigger to iterate through all the chart objects in `MultiChart`, and invoke the calculations for each. Each chart object calls a `PINQAnalyser` object with the query parameters, which in turn performs the given experiment – executing the query for each iteration and storing the results before passing it back to the chart object. An important part of this, which has a direct impact on the resulting chart, is how the bins are created (considering putting every value on the chart could be troublesome to read, especially with a high number of iterations); after all results have been generated, this array gets passed into a grouping algorithm, which takes the size of the chart (fixed to 100) and divides it by the number of bins requested. This results in a "bin size", which is a range as a subset of all the results. The data array is then iterated through and each value is copied to its respective bin (e.g. a value of 23 would go in the 3rd bin if 10 bins were used, this would correspond to bin 20-25 on the output chart). Once the calculations have been done, the chart controller invokes a chart object method to build the charts and send the `MultiChart` object back to the client. At this point the charts have been displayed on the client-side and the process is complete.

5.3. Testing

The site was tested before each commit to ensure it was properly working. However due to the experimental nature of the project, this couldn't always be the case, some situations arose where it seemed to be working as intended, but the maths behind were not operating properly.

Generally the site was tested each time according to the following requirements:

- Site loads correctly
- No CSS issues found

- No JavaScript errors when using developer console in browser
- Each link works correctly
- No runtime errors are encountered while adding, removing, or generating charts
- Charts generate correctly (see below)

The charting was tested by using a 'control' query, that is, a query that is used every time, meaning that any problems that have occurred within PINQ, or anything to do with the charting/computation of results, will reflect in the resulting chart. This query was usually the default – 1000 iterations, 10 bins, epsilon 1.0, Laplace distribution and average query, using the data {1,2,3,4,5,6,7,8,9,10} (these were scaled up by a factor of 10 when the results were clamped to 0 – 100).

This would produce a Laplace distribution centred on the middle of the chart – it was fairly easy to tell if something had gone wrong as the results would be wildly skewed.

5.4. Validation

Each parameter that is provided by the user must be validated to ensure that runtime errors are not encountered. The following validation regime was used to validate the data:

- Simple/Home Data
  - Iterations
    - Must be an integer
    - 0 > iterations > 100,000 (Hard limit)
  - Bins
    - Must be an integer
    - 0 > bins > 250 (Hard limit)
  - Epsilon
    - Must be a double
    - 0 > epsilon > Max(double)
  - Query type
    - Restricted to:
      - Average
      - Median
  - Timespan
    - Restricted to:
      - Hour
      - Week
      - Month
      - Year
  - Distribution
    - Restricted to:
      - Laplace
      - Gaussian
  - Delta
    - Must be a float
    - 0 > delta > 1
- Simple Data
  - Data list
    - Must be comma delimited
    - Must contain integers or doubles
    - E.g. "1,2,3,4,5"
    - Can contain spaces
- Home Data
  - Data category
    - Restricted to:
      - Temperature
      - Wind Speed
      - Humidity

The limits placed on iterations and bins are based on performance; the number of iterations directly affects the time it takes for a chart to generate, with 100,000 iterations taking over 2 minutes to complete. For the sake of accidental entry of large amounts of iterations, this was set as the limit for input. The amount of bins has an effect on how readable the chart is, as the bins approach 250, they begin to overlap one another horizontally, therefore vastly reducing the accuracy and readability of the chart.

5.5. Problems and difficulties

With the exception of the PINQ issues covered previously, several problems arose during the cycle between design and implementation. During the phase of importing the Smart* data into a database, it was realised that there were many missing values in the CSV (Column Separated Values) files which contained all the data. This wouldn't have been an issue, but the way SQL server bulk inserts, it needs to have a consistent number of columns; considering that the biggest CSV file was 650MB, this was a significant issue that would have taken forever to fix by hand – it was solved by the quick development of a tool that goes through CSV files and fixes the number of columns to a desired number, filling in the blanks with anything (zeros in this case). The source code for this is included with the project files.

A small issue that was encountered was to do with the version of the .NET framework supported by Arlia – it supports up to version 4, which meant the initial thoughts of using version 4.5 had to be scrapped. This meant excluding a simple threading mechanism using the '`async`' and '`await`' keywords provided by .NET v4.5, which make threading much easier in a web application.

Issues arose in the charting section of the site when it was realised that ASP.NET MVC didn't support the same charts as Windows Forms, this was resolved with the decision to use the Javascript library Highcharts. Unfortunately there was a problem with this – due to how the data was returned to the chart (results fell between two values, e.g. 20-25 – the result could be anything in between, the resolution of this is based on the number of bins selected), charts with a high number of bins would have an incredibly messy X axis, as Highcharts attempts to cram a value for each bin into it. The only solution that was found to this problem was to simply remove the X axis labels, considering that each bin is labelled with its respective value as you mouse over them. This resulted in a chart that allowed any number of bins with little effect on the performance or readability of the chart. The X axis has been left on the charts in the results sections for readability.

6. Results/Evaluation

6.1. Results

This section will cover the results of several queries using our site, full charts can be found in the report appendix.

The first result intends to show the drop-off zone between privacy and utility. For this set of queries, we used Laplace noise to obtain differentially private averages for the list: {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}; using various values of epsilon ranging from 0.001 to 10, we obtained the PINQ average 100 times, split into 20 bins, and took the most common value of the resulting chart (highest column in chart). If there were more than one most common value, the lowest of the set would be taken. Since the chart outputs values such as "20 - 25" (depending on number of bins), the lower of these two values were used – this shouldn't have any effect on the results. The experiment was run 5 times in order to gain a better idea of shape of the chart, and to remove the chance of a 'fluke' set of queries that, out of complete random luck, represented the results entirely differently to what it does most of the time.

Referring to Figure 3, which illustrates the value of the calculated average as epsilon rises, and shows each run as a different colour line; we can clearly see the point at where epsilon allows for a very accurate query.

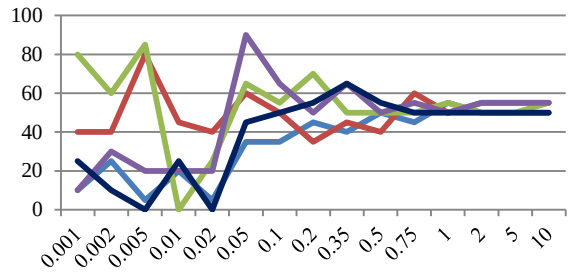

Figure 3: Averages vs. Epsilon

The actual average for the data list is 55, which we can see is being tended towards as epsilon increases. It seems that for this list of data the 'sweet spot', or the point that the queries move from noise to a reasonable result, is around epsilon 0.1. After this point, the results seem to only differ from the true average by around ±20. By the time epsilon reaches 0.75, the results differ by only ±10. At epsilon 2+, the query levels off at almost exactly 55, differing only by being at either 50 or 60, a difference of ±5 – the resolution of each bin.

The second result achieved is a version of the first example given in this report for differential privacy; the comparison of two identical sets of data, one of which has had a record removed. This experiment was done using the same data as above (a simple list of numbers); the parameters of this were 100 iterations with 20 bins, using an epsilon of 0.5 and the data list being: {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}, which, for the second part of the experiment, had the '100' removed to emulate a record (or individual) being removed.

Referring back to Section 2.3, where we describe differential privacy, we remember that the removal (or addition) of one record to the set of data can reveal information about that individual by comparing results with and without that record; we can protect against this by adding noise to the results. As shown in Figures 4 & 5, the site successfully hides the result of removing a record – minimal changes are made to the overall distribution, and one can imagine when given only one query (as opposed to the 100 for each chart here), you will be receiving incredibly similar values whether the value '100' is present or not.

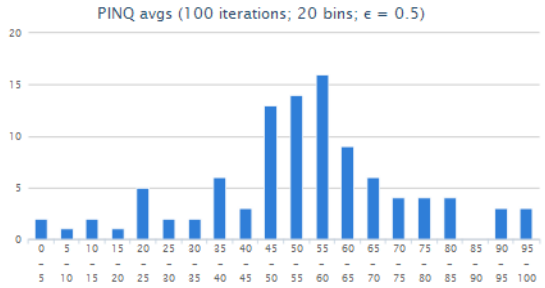

Figure 4: Simple data with all values included

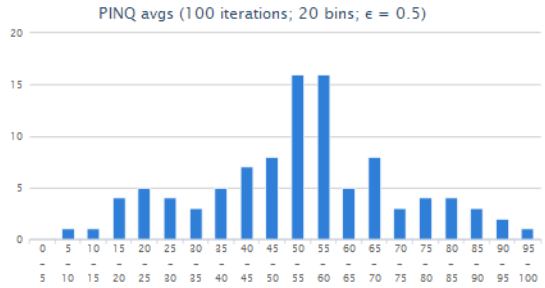

Figure 5: Simple data with value (100) removed

When investigating the home data, it was found to display only a singular bin when using a month's worth of data, as opposed to an hour (see Figure 6 & Figure 7, which shows the average temperature for both). It's important to note that the actual average of each is different – the average temperature of the hour is 28.408°F, and the average for a month is 62.179°F, meaning that the charts are centred at different points. However, every parameter is the same between them; the only difference is the amount of data. This observation confirms a fact about differential privacy, in that one needs to decide on the value of epsilon to use based on the data that it's being used against.

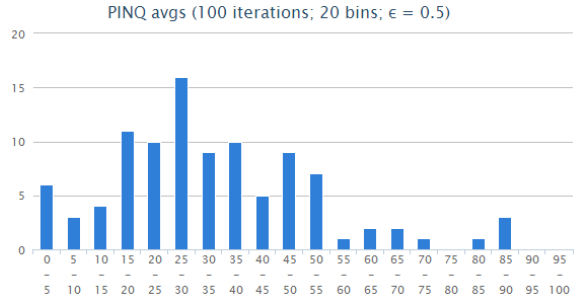

Figure 6: Home temperature data using an hour's worth of data

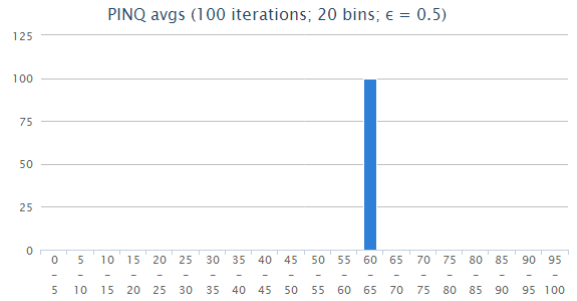

Figure 7: Home temperature data using a month's worth of data

The reason for this is fairly simple; the more data that is being used, the less impact noise has on the results. In the case of the hour of data, there are 11 data points, with the month there are 8754 – this means that during the average calculation, the tally (sum of all values) is very high, yet the amount of noise being added to each value is relatively small.

It seems that differential privacy is a good match for smart-home data; taking temperature as an example, using differential privacy means that the results for temperatures throughout the day are noisy enough so that it would be difficult to compare it against weather data accurately enough to locate the home.

Additionally, consider the case where a differentially private query is used to determine the average temperature inside the house each day, adding new data at midnight each night. Even if an attacker knows this fact, they may be able to compare the previous averages against the recently added one, but by adding noise to these queries, they will not be able to tell if the temperature has truly risen or lowered due to the probabilistic results, unless they are able to learn the noise added.

6.2. Evaluation

When comparing the final product to the initial requirements specification, the final product performs all the actions that it was initially intended to do. One notable difference is the exclusion of the ability to investigate the 'Power' field in the home data – this was down to the fact that values there extended the range supported by the graph (0 – 100), and despite efforts to think of another mechanism, it was not worth the effort to only include one extra metric.

It was decided, due mostly to time constraints, to scrap the idea of 'unknown' data – this was justified additionally by the fact it did not add much to the project, the same demonstrations (comparison of dataset and dataset minus 1) can be achieved by using the simple data charting on the site. One other removal from the final website, which was included in the design section of this report, is the clustering, which was intended to be used for complex queries that span a period of time, such as the average temperature every minute for an hour. The rationale behind the removal is again partially down to time constraints, and that using the complex queries that the clustering provided wasn't required to demonstrate differential privacy, and would have added a huge amount of complexity to the website.

Generally, as a platform for experimenting with differential privacy, our site works well to meet the requirements, allowing users to experiment with both basic and home data while performing average and median aggregate queries with a customisable epsilon, using a modified version of PINQ.

7. Appraisal

7.1. Critical Evaluation

In hindsight, there are a few things that would be done differently if this project were to be done again. The first of which is choosing a different platform on which to develop the site – MVC4 is a fantastically useful and powerful platform, however a large chunk of project time was spent learning how to use it, and being held up by misunderstandings of how it works. Perhaps the choice of using the more basic Web Forms, or even something like JSP would have been useful due to the prior knowledge of the platforms.

More time could have been dedicated to understanding how Highcharts work, in the end we had good charts being generated; however there still exists an issue with the X axis getting extremely congested when the number of bins starts to exceed 100 – perhaps a more elegant solution exists than just removing the labels.

In the end, the threading mechanism was not implemented in the final system due to time constraints, this could have been in place initially and been built around. This would be the preferred method if the project was to be done again, however, it was felt that this exclusion didn't take away from the project much, as it would simply be speeding the site up, and unless you're doing huge queries, you won't be waiting any longer than a few seconds.

7.2. Future Work

As far as future work is concerned, the website could be extended to provide further functionality; the ability to query any metric in the Smart* database, and the use of PINQ's transformation methods would make experimentation a lot more controllable.

The interface is not perfect currently, especially the data entry section that requires a comma-delimited list, is not particularly user-friendly and would benefit from an overhaul, perhaps using a completely separate dialogue box to setup various sets of data, of which could then be used to setup the queries. This would work well with another extension, which would be to implement a user system that allowed you to setup a profile and save interesting sets of data to it, and being able to recall them when creating charts (which could also be saved as results to the profile!). Another interface related improvement to the site would be a news/research section that provided a feed of current differential-privacy-related news (new articles/papers, etc.).

The architecture of the site could be upgraded, the exclusion of the threading mechanism meant that the site is slightly slower than it was designed to be – the addition of this would increase the speed at which results are returned to the user. Other optimisations could be performed, such as refactoring the code to remove unnecessary repeated chunks of code that may exist. These optimisations will be speed and efficiency oriented, and would not change how the site works.

Another extension could be to provide more complex aggregate queries for the user to include in their charts. PINQ has several more of these to use; data transformations were not used in this project, they could be included to modify the data further, and the use of PINQ's 'where' transformation could be used to replace the SQL currently used to retrieve sets of data from the database.

We have integrated PINQ with additional functionality to use either Laplace *or* Gaussian noise, and have altered the aggregate methods such that no work needs to be done outside of the method to cast values back to their original scale. The scale used by the method was set to between 0 and 100 – work needs done to figure out a more fluid way of doing this, for usage on values outside of that range. One suggestion would be to use a random value added onto the true maximum and minimum of the original range, however this still may be able to be learnt by repeated queries.

7.3. Knowledge/skills gained

A host of new skills have been acquired during the project, most notably an understanding of differential privacy and its role in protecting the privacy of data. The project also brought to light issues that had not previously been considered, such as the methods to ascertain disturbingly accurate information about data that one has no access to, and how important it is that this is prevented in as many cases as possible.

Knowledge and understanding of ASP.NET MVC is a valuable outcome to the project, while fairly difficult to understand after only using Web Forms, MVC seems a fantastic replacement for it – the ability to represent a page's data using a model will be incredibly useful for future projects that require ASP.NET/C#. On that note, C# skills were practiced thoroughly, which already was a favourite language and was great to work with.

A good amount of mathematics knowledge was gained during the project, due to the mathematical nature of differential privacy. Concepts such as distributions, which previously were acknowledged, are now understood to a fair degree, despite not having much of a mathematical background originally.

7.4. Acknowledgements

Sam would firstly like to thank his supervisor, Dr. Marco Gaboardi, for his guidance throughout the project, as well as for the project idea and support with understanding differential privacy. He would also like to thank his family and friends for their continued support during his time at university, and the School of Computing lecturers and other staff who provided the teaching and help required for a personal project of this scale.

8. Works Cited

Barker, S. et al., 2012. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. *ACM SustKDD'12.*

Dwork, C., 2008. Differential Privacy: A Survey of Results. *TAMC 2008.*

Dwork, C., 200X. A Firm Foundation for Private Data Analysis. pp. 1-8.

Dwork, C., Naor, M., Pitassi, T. & Rothblum, G. N., 2010. Differential Privacy Under Continual Observation. *STOC'10.*

Greveler, U., Justus, B. & Loehr, D., 20XX. Multimedia Content Identification Through Smart Meter Power Usage Profiles.

Highcharts, 2014. *Highcharts.* [Online] Available at: http://www.highcharts.com/ [Accessed March 2014].

McSherry, F., 2009. Privacy Integrated Queries. *SIGMOD '09.*

McSherry, F., 2010. Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis. *Communications of the ACM,* 53(9), pp. 89-97.

Molina-Markham, A. et al., 2010. Private Memoirs of a Smart Meter. *BuildSys.*

Qmee, 2013. *What Happens on the Internet in 60 Seconds.* [Online] Available at: http://blog.qmee.com/wp-content/uploads/2013/07/Qmee-Online-In-60-Seconds21.png [Accessed 9 April 2014].

Sweeney, L., 2002. k-Anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty.*

UMass, n.d. *Smart* Dataset.* [Online] Available at: http://traces.cs.umass.edu/index.php/Smart/Smart [Accessed January 2014].

Walker, C., 2013. Data Mining of Home Data Revealing Lifestyle Changes. *School of Computing Honours.*

Xu, J. et al., 2012. Differentially Private Histogram Publication. *2012 IEEE 28th International Conference on Data Engineering,* pp. 1-12.