Money for Nothing and Privacy for Free?

Swapneel Sheth, Tal Malkin, Gail Kaiser

Department of Computer Science, Columbia University, New York, NY 10027
{swapneel, tal, kaiser}@cs.columbia.edu

Abstract—Privacy in the context of ubiquitous social computing systems has become a major concern for the society at large. As the number of online social computing systems that collect user data grows, this privacy threat is further exacerbated. There has been some work (both, recent and older) on addressing these privacy concerns. These approaches typically require extra computational resources, which might be beneficial where privacy is concerned, but when dealing with Green Computing and sustainability, this is not a great option. Spending more computation time results in spending more energy and more resources that make the software system less sustainable. Ideally, what we would like are techniques for designing software systems that address these privacy concerns but which are also sustainable - systems where privacy could be achieved “for free,” i.e., without having to spend extra computational effort. In this paper, we describe how privacy can be achieved for free - an accidental and beneficial side effect of doing already existing computation - and what types of privacy threats it can mitigate. More precisely, we describe a “Privacy for Free” design pattern and show its feasibility, sustainability, and utility in building complex social computing systems.

Keywords—Design Pattern; Correlation Privacy; Web 2.0; Concept Drift; Differential Privacy;

I. INTRODUCTION

Today’s college students do not remember when social recommendations, such as those provided by Amazon, Netflix, Last.fm, and StumbleUpon, were not commonplace. Privacy in the context of these social computing systems has become a major concern for the society at large. A search for the pair of terms “facebook” and “privacy” gives nearly two billion hits on popular search engines. Recent feature enhancements and policy changes in social networking and recommender applications – as well as their increasingly common use – have exacerbated this issue [1]–[4]. With many online systems that range from providing purchasing recommendations to suggesting plausible friends, as well as media attention (e.g., the AOL anonymity-breaking incident reported by the New York Times [5]), both users of the systems and even non-users of the systems (e.g., friends, family, co-workers, etc. mentioned or photographed by users) are growing more and more concerned about their personal privacy [6].

Social computing systems, when treated in combination, have created a threat that we call “Correlation Privacy.” Narayanan and Shmatikov [7] demonstrated a relatively straightforward method to breach privacy and identify individuals by correlating anonymized Netflix movie rating data with public IMDb data. A similar de-anonymization approach could potentially be applied to any combination of such data-gathering systems, so how to safeguard against these “attacks” is an important concern for the designers of social computing systems. This is analogous to earlier work addressing queries on census data but, at that time, there were relatively few prospective attackers [8], [9].

There has been some recent work on data anonymization for privacy in software testing [10]–[12]. However, data anonymization alone may not be sufficient as Narayanan and Shmatikov show. (For more details on the related work including de-anonymization approaches, please see Section VI.) We need other techniques (which may be used orthogonal to data anonymization) to deal with privacy concerns and general approaches, design patterns, software architectures, etc. that would work across a wide variety of systems.

In this paper, we propose a design pattern, which we call “Privacy for Free,” targeted towards online social systems. In particular, we focus on systems that already have access to user data such as purchase history, movie ratings, music preferences, and friends and groups and that use complex data mining techniques for providing additional social benefits such as recommendations, top-n statistics, and so on to their users. In our software engineering community, these are systems like Mylyn [13], Codebook [14], or others([15], [16]) that have access to user (developer or end-user) interactions with software artifacts such as code, bug reports, and test cases.

The main research question we try to answer here is - Is there a general purpose architecture or design pattern that can be used with a wide range of large complex software systems, that will achieve privacy without spending any extra resources on computational overhead? We believe it is - we have discovered a technique for achieving privacy as an accidental and beneficial side effect of doing already existing computation.

The already existing computation in our case was weighing user data in a certain way - weighing recent user data exponentially more than older data to address the problem of “concept drift” [17] - to increase the relevance of the recommendations. This weighing is very common and used in a lot of systems [18]–[21]. Recent work in the databases/cs theory communities on Differential Privacy [22], [23] made us realize that our already existing computation for weighing user
data is very similar to one of the techniques for achieving differential privacy. (Intuitively, differential privacy ensures that a user’s participation (vs not participating) in a database doesn’t affect his privacy significantly. We provide more detailed information on Differential Privacy in Section III.) This resulted in the formulation of our hypothesis - if we change the existing computation so it matches the technique for achieving differential privacy (which would be a very minor and straightforward code change as the two techniques are very similar), would we get privacy as a beneficial side effect of addressing a completely different problem?

We show that it is indeed possible to get privacy as a beneficial side effect of doing some existing computation - thus, privacy for free - and this is the main contribution of our paper. We have formulated this technique as a design pattern that can be used in a wide variety of software systems to achieve "privacy for free", and show the feasibility, sustainability, and utility of using this approach to building software systems. We also contribute to the discussion in the privacy community about how to define privacy and how to achieve it. Specifically, we suggest a new direction for designing (differentially, or otherwise) private algorithms and systems motivated by what is already being done anyway.

There’s an added benefit of having privacy for free as a side-effect - even though privacy is important for users, many corporations may not be motivated to work hard on privacy. This may be due business reasons where having as much user information as possible is useful for targeted advertising, etc. If privacy could be achieved cheaply (in terms of computational or other resources), they still may not opt for it. In such cases, having privacy as a side effect of doing other computation is a very strong advantage from the users’ point of view.

The rest of the paper is organized as follows: Section II describes the motivation of our problem and why making privacy sustainable is important. Section III provides background information on Differential Privacy and Concept Drift. Section IV describes our “Privacy for Free” design pattern. Section V presents our empirical evaluations to show the feasibility, sustainability, and utility of our design pattern. Finally, we conclude the paper in Sections VI and VII with a discussion of the related work and our conclusions.

II. Motivation

Green Computing (or Green IT) is “the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems [...] efficiently and effectively with minimal or no impact on the environment” [24]. With our oil reserves projected to exhaust in less than fifty years [25], and renewable energy sources still providing only a small fraction [26], Green Computing here and now is becoming more and more important and, indeed, vital to our children and grandchildren.

An important research direction will be investigating how to build greener and more sustainable software systems from a software engineering perspective, in addition to the complementary algorithmic efficiency and systems perspective such as resource allocation, platform virtualization, and power management pursued by other computer science subdisciplines [27]. Ideally, from a sustainable software system point of view, we want to build systems that solve real-world problems by spending very little (or no) extra computational effort.

There has been some recent work in the software engineering community on data privacy [10]–[12]. This has focused on anonymization techniques to make hide sensitive data. While this work has been very promising, its goal hasn’t been to be sustainable. Clause and Orso [10] in their empirical results show that their technique takes between 2.5 minutes to 9 minutes. The time taken would probably increase for larger more complex systems. Similarly, the technique proposed in [11], [12] also requires substantial computation time. If there are millions of users of these systems, we are spending a lot of extra computational resources that aren’t needed as far the original system in concerned.

This is our main motivation for this paper. We feel that our “Privacy for Free” design pattern can result in software systems that are more sustainable and that already have privacy guarantees built in.

III. Background

Here we provide some background information on Differential Privacy and Concept Drift.

A. Differential Privacy

In the 1970s, when research into statistical databases was popular, Dalenius [28] proposed a desideratum for statistical database privacy - access to a statistical database should not enable someone to learn something about an individual that cannot be learned without access to the database. While such a desideratum would be great for privacy, Dwork et al. [22], [33] showed that this notion of absolute privacy is impossible using a strong mathematical proof. The problem with the desideratum is the presence of “Auxiliary Information”. Auxiliary Information is similar to, and a generalization of, the notion of Correlation Privacy mentioned earlier.

Dwork gives a nice example to explain how Auxiliary Information can be a problem when privacy is concerned - “Suppose one’s exact height were considered a highly sensitive piece of information, and that revealing the exact height of an individual were a privacy breach. Assume that the database yields the average heights of women of different nationalities. An adversary who has access to the statistical database and the auxiliary information “Terry Gross is two
inches shorter than the average Lithuanian woman” learns Terry Gross’ height, while anyone learning only the auxiliary information, without access to the average heights, learns relatively little.” An interesting observation made by Dwork is that the above example for breach of privacy holds regardless of whether Terry Gross’ information is part of the database or not.

To combat Auxiliary Information, Dwork proposes a new notion of privacy called Differential Privacy. Dwork’s paper is a culmination of the work started earlier and described in papers such as [29]–[31]. Intuitively, Differential Privacy guarantees privacy by saying that if an individual participates in the database, there is no additional loss of privacy (within a small factor) versus if he had not participated in the database. Formally, Differential Privacy is defined as follows: A Randomized function \( K \) gives \( \epsilon \)-differential privacy if for all data sets \( D_1 \) and \( D_2 \) differing on at most one element, and all \( S \subseteq \text{Range}(K) \),

\[
\Pr[K(D_1) \in S] \leq \exp(\epsilon) \times \Pr[K(D_2) \in S]
\]  

C. Concept Drift

People’s preferences change over time - things that I like doing today may not be things I liked doing 10 years ago. If data is being mined or recommendations being generated, the age of the data needs to be accounted for. To address this problem, the notion of Concept Drift was formed [17]. This problem needs to be addressed by any field that deals with data spanning some time frame (from a few hours to months and years). An example class of systems that need to address the problem on Concept Drift is Recommender Systems. Many recommender systems use Collaborative Filtering (CF), i.e., recommending things to an individual by looking at what other users similar to the individual like [21], [36], [37]. CF algorithms typically look at the activities of individuals from the past (movies watched, things bought, etc.) and use this to derive recommendations. However, people’s preferences change over time. For example, when I am in college and taking a lot of classes, I might buy a lot of textbooks from Amazon. When I graduate, I may not need textbook recommendations. This is exactly the kind of problem that Concept Drift tries to address.

Other example classes of systems that need to address this problem are systems that mine software repositories [38], social software engineering systems [14], systems for collaboration and awareness [39], etc. For these kinds of systems, there is a lot of old and recent data available and weighing certain data differently might be essential.

D. Addressing Concept Drift

There have been different solutions proposed to address the problem of Concept Drift [17], [40], [41]. A particular solution of note is the Exponential Time Decay Algorithm [42]. The Exponential Time Decay Algorithm weighs things
done recently exponentially higher than things done in the past. It gradually decays the weight of things done in the past so that things done in the distant past do not affect the outcome as much as things done recently, thus addressing the problem on Concept Drift. The Exponential Time Decay Algorithm is very popular and used by a lot of systems [18]–[21]. For the rest of the paper, we refer to this as the CD (Concept Drift) algorithm.

Consider the Terry Gross example again and let’s assume that the database has historical data going back 100 years. As average heights change over time, the CD algorithm will weigh newer data exponentially higher than older data resulting in a weighted average height. This would reflect the recent trends but also account for older data. The CD algorithm is the another corner stone of our design pattern and we build on it more in the next section.

IV. PRIVACY FOR FREE: A DESIGN PATTERN

The CD and EM algorithms are very similar. The CD algorithm uses exponential weighing over the data while the EM algorithm chooses inputs with probability proportional to the exponential of the scoring function. If we choose the scoring function to be the timestamp of the data, the two algorithms becomes even more similar. The CD algorithm is deterministic and weighs new data exponentially higher than older data; the EM algorithm is probabilistic and chooses new data with an exponentially higher probability than older data.

This is the crux of our paper - if existing systems that already use the CD algorithm modify the code to use the EM algorithm instead, they would, as an added benefit, get the main advantage of the EM algorithm - differential privacy. Further, this privacy would not require any extra computational overhead and thus, we would get privacy for free. Systems that do not already use either the CD or the EM algorithm could still add the EM algorithm and privacy could still be viewed as being free - an added benefit of solving some other problem, which in this case is Concept Drift. Since these two algorithms are very similar, it would require a very small and straightforward change to the code to change from the CD algorithm to the EM algorithm.

The important requirement for the differential privacy guarantees to hold are that all the data access must be done via the EM algorithm, which could be implemented as a separate class or be part of a library or the data model, etc. We describe our design pattern using a modified version of the template suggested by Gamma et al. [43]. This is shown below:

Pattern Name and Classification
“Privacy for Free”, Behavioral class pattern

Intent
To provide differential privacy in social computing systems without any extra computational overhead.

Motivation
See Section II.

Applicability
Software systems that already have access to user data such as purchase history, movie preferences, interaction with software artifacts like code and bug reports.

Participants
The rest of the design of an existing system can remain unchanged. A new system can be implemented as per the necessary requirements. The only mandatory class is the EM algorithm and this must be used to access the data. There will be no other change in participants.

Collaborations
There are no requirements on participant collaborations. The only restriction is that all access to the data should be via the EM algorithm, which could be implemented as a class (or part of a library or data model).

Consequences
The design pattern will provide privacy for free without any extra computational overhead. The tradeoff is a small loss in accuracy of recommendations/data mining. See Section V-C.

Implementation
Any programming language can be used for the implementation. The only requirement would be the ability to generate pseudo-random numbers as the algorithm is probabilistic.

Related Patterns
None as we focus on systems that already have access to user data. For other kinds of systems (such as network systems), there are existing privacy patterns [44], [45].

V. EVALUATION

Our design pattern requires implementing (or substituting an existing implementation of the CD algorithm with) the EM algorithm. To evaluate our design pattern, we implemented the EM and CD algorithms and investigated the differences in these. Our goal was to answer the following research questions:

RQ1: Feasibility—Does using our design pattern guarantee differential privacy?
RQ2: Utility—Does using our design pattern affect the utility of the system to give meaningful recommendations or mine data?
RQ3: Sustainability—Can our design pattern be sustainable? Can using our design pattern result in no additional computational resources for privacy?
With RQ1, we aim to prove the primary benefit of our design pattern - guaranteeing privacy. Our goal is to show that it does indeed guarantee differential privacy making it suitable to be used in a variety of large social systems.

With RQ2, we explore the utility of using our design pattern. A “straw man” way to guarantee privacy for any recommender/data mining system is to give a random answer every time. This would not require any clever technical solutions, but this would be very bad for the overall utility of the system - the goal of most such systems is to provide relevant information. There exists a tradeoff between accuracy and privacy and we explore this here. We aim to show that, using our technique, there is a small loss in accuracy and that this loss in accuracy scales very well (roughly constant) as the size of the system increases. Thus, if a small loss in accuracy is acceptable, we can get privacy for free without spending any additional computational resources.

With RQ3, we aim to show the sustainability benefits of using our design pattern. We show that using our design pattern (and the EM algorithm) requires less CPU time than the equivalent CD algorithm. Not only do we not need any additional computational resources, we should be able to reduce computational needs by using our design pattern.

A. RQ1 - Feasibility

Our design pattern requires the use of the EM algorithm for all access to the data. The EM algorithm that we require is exactly the same as the one proposed by McSherry and Talwar [23]. The algorithm they propose can work with different scoring functions that weigh the data differently - in our case, the scoring function we use is the timestamp of the data. Our use of the EM algorithm in our design pattern can thus be viewed as an instantiation of the general EM algorithm. McSherry and Talwar show a theoretical proof for the EM algorithm to be differentially private. We do not repeat the proof here and we encourage the interested reader to look at the paper (page 5 of [23]). As all data access happens via the EM algorithm, our design pattern also guarantees differential privacy.

B. Methodology

For RQ2 and RQ3, we carried out experiments to validate our hypotheses. We use synthetic data as there are no benefits of using real world data for our hypotheses. We create an array of size $n$ and randomly fill it with values from 0 to $n - 1$. Each element has a timestamp associated with it to simulate user activity - for the purpose of this experiment, we assume that the timestamp is the array index. A lower array index indicates that the item is newer. Thus, we want to prefer items with a lower index in the output as these items indicate things that are done recently.

Using the differential privacy EM algorithm [23], we choose the scoring function to be maximized by returning a value with as low an array index as possible. Thus, we choose elements from the array with probability based on their array index.

In the experiments, we randomly generate the array and compute the score using the CD and the EM algorithms. We then plot the RMS and normalized RMS errors between these two algorithms. We discuss the results in the following subsections.

C. RQ2 - Utility

For the first set of experiments, we varied the size of the array and plotted the RMS and normalized RMS errors between the CD and EM algorithms. The results are shown in Figure 1. To smooth out the noise in the experimental results (as CD is a deterministic algorithm while EM is a probabilistic one), we ran the experiment 1000 times with each array size and took averages. The graph shows us that as the size of the input array increases, the RMS error increases linearly - this is expected as with larger array sizes, the entries in the array have correspondingly larger values (due to our methodology), resulting in linearly increasing RMS error. Meanwhile, the normalized RMS error is roughly constant.

This shows us the tradeoff between accuracy and privacy. We observe that in these experiments, the loss of accuracy is relatively small - the normalized RMS error is less than 0.4. Thus, irrespective of the data set size, switching to the EM Algorithm (as required by our design pattern) from the CD Algorithm will not worsen the accuracy of the algorithm by more than the constant factor, and we have the added benefit that the EM algorithm also guarantees differential privacy. Whether the loss of accuracy is acceptable or not (or a worthy price to pay for the free privacy) is subjective and we deliberately do not enter a philosophical debate here.

For our second set of experiments, we varied the number of trials keeping the size of the array fixed to 1000. The graph plotting the RMS error vs the number of trials is shown in Figure 2. This graph shows us that as the number of trials increases, the RMS error reduces. Thus, initially,
even though there may be a bigger error between the CD and EM algorithms, in the long run, the error will be small.

With these set of experiments, we explored the utility of our design pattern. For an existing system (that may already use an algorithm similar to the CD one), a one-time change would be required to add in the EM algorithm and retrofit the system to our design pattern. This change is relatively straightforward and could even be automated. Making such a change, albeit results in a small loss of accuracy, gives the huge benefit of getting privacy for free without spending any additional computational resources.

D. RQ3 - Sustainability

For RQ3, we want to show the sustainability of our design pattern. With the EM algorithm in place, what we ideally want is that our system does not take any additional computational resources. We decided to use the CPU processing time to estimate the computational resources needed by the two algorithms. We instrumented the CD and EM algorithms and measured how long they took in the first set of experiments in Section V-C above. The resultant graph is shown in Figure 3. The graph shows us that for all data sizes the EM algorithm took less CPU time than the CD algorithm.

Not only does the EM algorithm not require any additional computational resources, it actually reduces the existing computation. Thus, changing to our design pattern will make the software system even more sustainable.

E. Threats to Validity

The notion of Differential Privacy may not relate to the user-centric view of Privacy as users might think it “strange” that the system assumes that bad things can happen anyway - the guarantee it gives is just regarding whether the user data is part of the system or not. While that is true, we feel that differential privacy has many compelling arguments in its favor - the biggest, for us, is not having to decide what data is sensitive and what is not. The differential privacy algorithms treat all data as sensitive making it easier not to leak data by accident. One would, therefore, not have to deal with the subjective nature of deciding what’s sensitive. We also feel that the guarantee might actually make it even more compelling for the user. From their point of view - “if bad things are going to happen anyway, it’s not going to hurt me much more if I participate.. so there’s no harm in participating.”

We used synthetic data in our evaluations rather than real-world data. For the research questions that we had - feasibility, utility, sustainability - synthetic data was sufficient. The only benefit of using real world data would be to answer some other research questions that are outside the scope of this paper.

VI. RELATED WORK

There have been some recent papers on data privacy and software testing. Clause and Orso [10], proposes techniques for the automated anonymization of field data for software testing. They extend the work done by Castro et al. [46] using novel concepts of path condition relaxation and breakable input conditions resulting in improving the effectiveness of input anonymization. Our work is orthogonal to the papers on input anonymization. The problem they address is - how can users anonymize sensitive information before sending it to the teams or companies that build the software? The problem we address is - how can systems that already have access to user data (such as purchase history, movie preferences, and so on) be engineered so that they don’t leak sensitive information while doing data mining on the data? Further, the aim of our approach is to provide privacy “for free,” i.e., without spending extra computational resources on privacy. The input anonymization approaches require spending extra computation as they address a different problem. We believe that the our approach can be combined with the input anonymization approach if needed based on user needs. If users are worried about developers at the company finding out sensitive information, input anonymization is essential. If, however, they are worried about accidental data leakage through the data mining of their information, using
the “Privacy for Free” design pattern may be more suitable. This would also make the software system more sustainable as we don’t spend any computation doing the anonymization of the inputs.

Taneja et al. [11] and Grechanik et al. [12] propose using k-anonymity [47] for privacy by selectively anonymizing certain attributes of a database for software testing. Their papers propose novel approaches using static analysis for selecting which attributes to anonymize so that test coverage remains high. Similar to above, our approach is orthogonal as we focus on a design pattern that will prevent accidental leakage of sensitive information via data mining or similar techniques. Further, these approaches using k-anonymity also require significant additional computational resources and thus, may not be sustainable when energy resources are scarce.

Our differential privacy approach has the added benefit of being able to work with any kind of data and not being limited to just integers or such. Finally, work on input anonymization and k-anonymization both focus on software testing whereas our approach focuses on a design pattern for building privacy preserving systems with a specific goal - to make privacy sustainable and not require additional resources.

There has also been a lot of work related to data anonymization and building accurate data models for statistical use (e.g., [48]–[52]). These techniques aim to preserve certain properties of the data (e.g., statistical properties like average) so they can be useful in data mining while trying to preserve privacy of individual records. The broad approaches include aggregating data to a higher level of granularity or adding noise and random perturbations. As we are interested in sustainable ways of achieving privacy, these approaches are not applicable as they typically require (a lot of) extra computational effort.

While there has been a lot of interest (and research) in data anonymization, we would like to reiterate that only data anonymization might not be enough. Narayanan and Shmatikov [7] demonstrate a relatively straightforward way of breaking the anonymity of data. They show how it is possible to correlate public IMDb data with private anonymized Netflix movie rating data resulting in the potential identifi-
cation of the anonymized individuals. Backstrom et al. [53] also describe a series of attacks for de-anonymizing social networks that have been anonymized to be made available to the public. They describe two categories of attacks - active attacks where an evil adversary targets an arbitrary set of users and passive attacks where existing users try to discover their location in the network and thereby cause de-
anonymization. Their results show that, with high probability and modest computational requirements, de-anonymization is possible for a real world social network (in their case, LiveJournal [54]).

VII. Conclusion

As social computing systems that collect users’ data proliferate, privacy has and will continue to become a major concern for the society at large. The main research question that we wanted to answer is - Is there a general purpose architecture or design pattern that can be used with a wide range of large complex software systems, that will achieve privacy without spending any extra resources on computa-
tional overhead? Our “Privacy for Free” design pattern can achieve privacy as an accidental and beneficial side effect of doing some existing computation. The results of our evaluations show the feasibility, utility, and in particular, the sustainability of our approach as it does not require any additional computational resources to guarantee privacy.

Acknowledgment

Sheth and Kaiser are members of the Programming Sys-
tems Lab, funded in part by NSF CNS-0717544, CNS-
0627473 and CNS-0426623, and NIH 2 U54 CA121852-06. Malkin is a member of the Crypto Lab, funded in part by NSF 0831094 and 0347839 and DHS N66001-09-C-0080.

References

stm, July 2010.


