The Boundary Between Privacy and Utility in Data Publishing[*]

Vibhor Rastogi Dan Suciu Sungho Hong

ABSTRACT

We consider the privacy problem in data publishing: given a database instance containing sensitive information "anonymize" it to obtain a view such that, on one hand attackers cannot learn any sensitive information from the view, and on the other hand legitimate users can use it to compute useful statistics. These are conflicting goals. In this paper we prove an almost crisp separation of the case when a useful anonymization algorithm is possible from when it is not, based on the attacker's prior knowledge. Our definition of privacy is derived from existing literature and relates the attacker's prior belief for a given tuple t, with the posterior belief for the same tuple. Our definition of utility is based on the error bound on the estimates of counting queries. The main result has two parts. First we show that if the prior beliefs for some tuples are large then there exists no useful anonymization algorithm. Second, we show that when the prior is bounded for all tuples then there exists an anonymization algorithm that is both private and useful. The anonymization algorithm that forms our positive result is novel, and improves the privacy/utility tradeoff of previously known algorithms with privacy/utility guarantees such as FRAPP.

1. INTRODUCTION

The need to preserve private information while publishing data for statistical processing is a widespread problem. By studying medical data, consumer data, or insurance data, analysts can often derive very valuable statistical facts, sometimes benefiting the society at large, but the concerns about individual privacy prevents the dissemination of such databases.

Clearly, any anonymization method needs to trade off between privacy and utility: removing all items from the database achieves perfect privacy, but total uselessness, while publishing the entire data unaltered is at the other extreme. In this paper we study the tradeoff between the privacy and the utility of any anonymization method, as a function of the attacker's background knowledge. When the attacker has too much knowledge, we show that no anonymization method can achieve both. We therefore propose to study anonymization methods that can protect against bounded adversary, which have some, but only limited knowledge. We show that in this case the tradeoff can be achieved.

Age	Nationality	Score
25	British	99
27	British	97
21	Indian	82
32	Indian	90
33	American	94
36	American	94

(a) Test Scores

Age	Nationality	Score
21-30	*	99
21-30	*	97
21-30	*	82
31-40	*	90
31-40	*	94
31-40	*	94

(b) 2-diversity and 3-anonymity [13]

Age	Nationality	Score
25	**British**	**99**
28	Indian	99
29	American	81
32	**Indian**	**90**
39	American	84
32	Indian	89

(c) FRAPP [3]

Age	Nationality	Score
25	**British**	**99**
21	British	99
22	Indian	89
32	**Indian**	**90**
28	Indian	99
29	American	81
33	**American**	**94**
27	American	94
32	British	83
36	**American**	**94**
26	American	99
39	Indian	94

(d) $\alpha\beta$ algorithm

Table 1: Different data publishing methods

1.1 Basic Concepts

Throughout the paper we will denote with I the database instance containing the private data. I is a collection of records, and we denote $n = |I|$ its cardinality. We also denote D the domain of all possible tuples, i.e. $I \subseteq D$ and denote $m = |D|$. For example, the data instance I in Fig 1 (a) consists of $n = 6$ records representing test scores, and D is the domain of all triples (x, y, z) where $x \in [16, 35]$ is an age, y is a nationality (say, from a list of 50 valid nationalities), and $z \in [1, 100]$ is a test score, thus $m = 20 \cdot 50 \cdot 100 = 100,000$. The goal of an anonymization algorithm is to compute a new data instance V from I that offers both privacy (hides the tuples in I) and utility. The notion of

[*]This work was partially supported by NSF Grants IIS-0415193, IIS-0627585, and ITR IIS-0428168

VLDB '07, September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

	Definition	Meaning
D	Domain of tuples	
I	Database instance	$I \subseteq D$
V	Published view	$V \subseteq D$
n	Database size	$n = \|I\|$
m	Domain size	$m = \|D\|$, $n \ll m$
k	Attacker's prior	prior of tuple t, $Pr[t] \leq k\frac{n}{m}$
γ	Attacker's posterior	posterior $Pr[t\|V] \leq \gamma$
ρ	Query estimation error	$\|Q(I) - \tilde{Q}(V)\| \leq \rho\sqrt{n}$

Table 2: Symbols used in the paper; formal definitions in Sec. 2.

privacy that we use in this paper compares the attacker's prior probability of a tuple belonging to I, $Pr[t] = Pr[t \in I]$ with the a posteriori probability, $Pr[t|V] = Pr[t \in I|V]$. If the algorithm is private, then in particular: if $Pr[t]$ is low, $Pr[t|V]$ must also be low. It makes sense to express the attacker's prior as $Pr[t] = k\frac{n}{m}$, for some k, since the database contains n tuples out of m. We also denote γ a bound on the a posteriori. The notion of *utility* that we use in this paper measures how well a user can estimate counting queries. An example of a counting query Q for the data in Fig 1 (a) is *count the number people from poor African countries that received a score* > 98: assuming the user has a list of poor African countries, the answer of Q on I, denoted $Q(I)$, can simply be obtained by scanning the table I and counting how many records satisfy the predicate. But the user doesn't have access to I, only to V, hence she will compute instead an estimate, $\tilde{Q}(V)$, over V. Our definition of utility consists in a guaranteed upper bound on the absolute error of the estimator, more precisely, it is given by a parameter ρ s.t. $|Q(I) - \tilde{Q}(V)| \leq \rho\sqrt{n}$. (We explain the particular choice of $\rho\sqrt{n}$ in a moment.) These parameters are summarized in Table 2, and defined formally in Sec. 2.

1.2 Our Main Results

In this paper we prove two complementary main results. *First*, when $k = \Omega(\sqrt{m})$ then no algorithm can achieve both utility and privacy (Theorem 3.3). In other words, if the adversary is powerful, i.e. his prior is $Pr[t] = k\frac{n}{m} = \Omega(\frac{n}{\sqrt{m}})$ for some tuples t, then no algorithm can achieve both privacy and utility. This justifies us to consider adversaries that are bounded in power, i.e. their prior is bounded. *Second*, when $k = O(1)$, equivalently when the adversary's prior is bounded by $Pr[t] \leq k\frac{n}{m} = O(\frac{n}{m})$, then we describe an algorithm that achieves a utility of $\rho = \sqrt{k/\gamma}$ (Theorems 4.3 & 4.4). Here, and throughout the paper, the notations $O(-)$ or $\Omega(-)$ refer to asymptotic functions in n and m.

1.3 Related Work

We discuss now the related work in order to place our results in context. Privacy mechanisms can be classified into three categories, according to where they are deployed during the life cycle of the data since the time it is collected from individuals to the point it is made available for users. See Figure 1. In *local perturbation*, (1), the individuals trust no one but themselves, and they anonymize their respective data before transferring it to the central server. In *data publishing*, (2), anonymization occurs at the central server. Now the individuals are required to trust this server, but the advantage is that one may design better anonymization algorithms. All results in this paper are for data publishing. In *output perturbation*, (3), the data is kept on a trusted central server that accepts user queries, evaluates them locally, then returns a perturbed answer. The best privacy/utility tradeoffs can be achieved in this setting, but the cost is that the user can never see the data directly, and can access it only through the interaction with the server, which often is limited in what kinds of queries it supports. Some output perturbation algorithms place a bound on the number of queries, i.e. the server counts how many queries it answers on behalf of a user and stops once a limit has been reached. Referring to Fig. 1 we note that all "positive" results extend automatically to the right (e.g. any local perturbation algorithm can also be used in data publishing), while all "negative" results extend to the left.

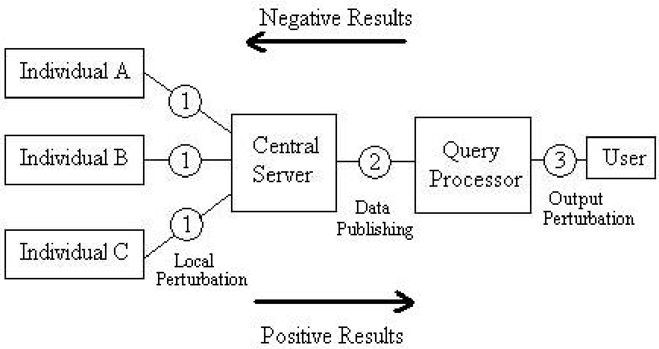


Figure 1: Data anonymization: (1) Local perturbation (2) Data publishing (3) Output perturbation.

The relevant results from the related work in the literature are captured in Table 3, where we attempted to present them in a unified framework. We explain now the positive and negative results.

Positive results There are several randomized algorithms for *local perturbation* [1, 9, 10, 18, 2, 15, 3]. Each tuple in the database I is locally perturbed by replacing it with a randomly chosen tuple according to a predefined probability distribution. FRAPP [3] is a framework that generalizes all algorithms in this class, and also achieves the (provably) best privacy/utility in this class: $\rho = k/\gamma$. Note that our algorithm improves this, by reducing ρ to $\rho = \sqrt{k/\gamma}$ (note that $k/\gamma > 1$ because $k > 1$ and $\gamma < 1$). The reason we could improve the utility is that we use the power of the trusted server to make better perturbation in the data. In *data publishing*, virtually all techniques described in the literature are variations of the k-anonymity method (e.g. [19, 13, 17, 20]), which do not guarantee the privacy and/or utility, and, hence, do not relate directly to our results. In *output perturbation*, a much better tradeoff of $\rho = log(k/\gamma)/\sqrt{n}$ can be achieved [8], if the number of queries can be bounded to $O(n)$. In contrast, our algorithm guarantees utility for all queries.

Negative results The first impossibility result was shown in [6] and says that any algorithm which guarantees $\rho = o(1)$ for more than $\Omega(nlog^2n)$ queries will be completely non-private. Their result is stated to say that no algorithm can achieve absolute errors $|Q(I) - \tilde{Q}(V)| \leq o(\sqrt{n})$, which corresponds to $\rho = o(1)$. The error guarantees in our algorithm

	Local perturbation (1)	Data publishing (2)	Output perturbation (3)	
			Unbounded queries	Bounded queries
Positive Results	FRAPP [3]: w.h.p. $\rho = \frac{k}{\gamma}$ [1] [9] [10], MASK [18], OLAP [2], Sketches [15]: weaker utility	**This paper**: w.h.p. $\rho = \sqrt{\frac{k}{\gamma}}$ (Theorems 4.3 & 4.4) k-anonymity [19], l-diversity [13], Swapping [17], Anatomy [20]: no formal guarantees		w.h.p $\rho = \frac{log(k/\gamma)}{\sqrt{n}}$, for O(n) queries SuLQ [5], Differential Privacy [7] [8]
Negative Results (utility)		**This paper**: If $\rho = o(1)$ with probability $> 1/2$, then no privacy (Theorem 2.6)	If $\rho = o(\sqrt{n})$ for all queries, then no privacy [6]	If $\rho = o(1)$ for $\Omega(nlog^2n)$ queries, then no privacy [6]
Negative Results (privacy)		**This paper**: Utility impossible for $k = \Omega(\sqrt{m})$ (Theorem 3.3)		

Table 3: Classification of privacy preserving techniques; w.h.p = with high probability; o(.) is used w.r.t n

are indeed larger, namely $\rho = O(1)$, but on the other hand we guarantee this for all queries. The second result in [6] is that no privacy can be guaranteed if all queries can be estimated with $\rho = o(\sqrt{n})$. This seems to contradict our result, which states $\rho = O(1) = o(\sqrt{n})$. The distinction lies in the fact that our error guarantee is only *with high probability* (denoted w.h.p. in Table 3) while the negative result in [6] makes the assumption that all queries are guaranteed to have that error bound. We explain now what we mean by "with high probability" (which is the same as in FRAPP). Our anonymization algorithm is a randomized algorithm s.t. for any query Q, the probability that the algorithm outputs a view V s.t. $|Q(I) - \tilde{Q}(V)| \leq \rho\sqrt{n}$ is $1 - \varepsilon$, where ε is a parameter of the algorithm. Now suppose we run the attack in [6] to our algorithm. In that attack the adversary evaluates a large number of counting queries, say N, and must obtain good estimates on all queries. Now, in order to attack our algorithm, all these queries are estimated over the same view V, so the probability (over V) that all estimates are good is $(1 - \varepsilon)^N$. When N is too large, the attack cannot continue: this is why the result in [6] does not contradict ours. On the other hand, if $N = \Omega(n \log^2 n)$, then we show (in Theorem 2.6) that the attack succeeds, hence no privacy/utility can be obtained in Data Publishing when $\rho = o(1)$ (see Table 3), even when $\varepsilon = 1/2$.

The negative results in [6] were obtained without giving any definition of an adversary and of privacy. They state that if one wants to have a lot of utility, then there is a major privacy breach that would violate whatever definition of privacy one adopts. Thus, our main impossibility result (Theorem 3.3) is incomparable to [6], and in some sense much more subtle. It states that if the adversary is more powerful than a certain bound, then we cannot achieve both privacy and utility. Our result is the first negative result trading off an adversary's power for utility.

The techniques that we use to derive our Theorem 3.3 improve over an impossibility result in [8] for a different notion of privacy, called *ϵ-indistinguishability*, which is stated for a different setting (without an adversary).

1.4 Example

Our anonymization algorithm is closest in spirit to, and improves over FRAPP, and we illustrate here the two algorithm's using the data in Table 1(a) (a database of English test scores).

FRAPP anonymizes data as follows. Independently for each tuple in the database I the algorithm randomly chooses whether to keep it in V, or whether to replace it with another tuple, randomly chosen from the domain D. The proportion of retained tuples is governed by a parameter γ_{frapp}. For example if we choose the parameter $\gamma_{frapp} = 1/3$, the a posteriori probability for the adversary with no prior knowledge remains less than 1/3. Table 1(c) illustrates a possible output of FRAPP. Two tuples from the original instance have been retained while four have been replaced by uniformly chosen random tuples. For illustration purposes the tuples that have been retained are highlighted.

Our algorithm, called $\alpha\beta$ operates in two steps. First, for each tuple in the instance I it randomly chooses whether to keep it in V or whether to remove it. Second, for each tuple in the domain D it randomly chooses to insert it in V with a probability β. A particular output of the algorithm is shown in Table 1(d): four tuples have been retained from the original data, and eight new tuples have been inserted. We highlight the four tuples that come from I. We choose the parameters α and β so that the a posteriori is same 1/3 as that of Table 1(c): this can seen by noting that Table 1(d) contains the same ratio 1/3 of true tuples to noise tuples. Thus, FRAPP and $\alpha\beta$ have the same privacy in this example. However, $\alpha\beta$ has a better utility, intuitively because it contains four of the original tuples, while FRAPP only retained two. We will prove this formally in Section 4.

1.5 Specific Contributions

In this paper we make the following specific contributions:

- We define a *bounded adversary*, a notion of *privacy* that protects against a bounded adversary, and a notion of *utility* that gives guarantees on the estimation of arbitrary counting queries. (Sec. 2.)
- We prove that for powerful adversaries no algorithm can achieve both privacy and utility (Sec. 3).
- We describe the $\alpha\beta$ anonymization algorithm that guarantees privacy and utility for bounded adversaries, and improves the utility of FRAPP. We describe an estimation algorithm for counting queries with arbitrary complex predicates. (Sec. 4).
- We present two extensions of the $\alpha\beta$ algorithm. First, we show how one can publish multiple views over the

same data, for example when the data has been updated. Then we discuss how one can further improve the privacy if one has some information about the adversary's prior. (Sec. 4.4)

- We validate experimentally the utility of our algorithm on a real dataset. (Sec. 5).
- We present an extension of our theoretical results to priors with correlations between tuples. (Sec. 6).

2. NOTATIONS AND DEFINITIONS

A database instance is a table consisting of n tuples, $I = \{t_1, t_2, .., t_n\}$, where each tuple has l attributes $A_1, \ldots, A_l$, and each attribute A_i takes values from a finite domain D_i. We denote $D = D_1 \times \ldots \times D_l$ the domain of all tuples with attributes $A_1, \ldots, A_l$, thus, $I \subseteq D$, and we write $m = |D|$. Our goal is to design an algorithm that computes a view $V \subseteq D$ from I s.t. (1) an adversary cannot determine, by using V, whether any particular tuple t belongs to I, and (2) any user can use V to compute approximate values to counting queries over I. The first property is *privacy*, the second is *utility*.

Modeling the Adversary We model the adversary's background knowledge using a probability distribution Pr_1 over the set of all possible database instances I. Formally, $Pr_1 : 2^D \to [0, 1]$ is such that $\sum_{I \subseteq D} Pr_1[I] = 1$. For each tuple $t \in D$ we denote $Pr_1[t] = Pr_1[t \in I]$ the marginal distribution (i.e. $Pr_1[t] = \sum_{I:t \in I} Pr_1[I]$), and call Pr_1 *tuple-independent* when the tuples are independent of each other. Clearly, we do not know Pr_1, only the attacker knows it, and we should design our privacy algorithm assuming the worst about Pr_1. As we shall see, however, it is impossible to achieve privacy and utility when the attacker is all powerful, hence we will make reasonable assumptions about Pr_1 below.

Modeling the Algorithm A privacy-preserving algorithm is a randomized algorithm A that, given I, computes a view V. We denote $Pr_2^I[V]$ the probability distribution on the algorithm's outputs, i.e. $Pr_2^I : 2^D \to [0, 1]$ is s.t. $\sum_{V \subseteq D} Pr_2^I[V] = 1$. Unlike Pr_1, we have total control over Pr_2, since we design the algorithm A.

As the algorithm A needs to be public, an adversary can compute the induced probability $Pr_{12}[t|V] = Pr_{12}[t \in I|V]$ based on his prior distribution Pr_1 and the algorithm's distribution Pr_2, namely $Pr_{12}[t|V] = \sum_{I \subseteq D:t \in I} Pr_2^I[V] Pr_1[I]$. We call $Pr_1[t]$ the prior probability of the tuple t and $Pr_{12}[t|V]$ the posterior probability of the tuple t conditioned on the view V. Throughout this paper we assume that the adversary is computationally powerful and that $Pr_{12}[t|V]$ can always be computed by the adversary irrespective of the computation effort required.

2.1 Classification of Adversaries

We will prove in Sec. 3 that it is impossible to achieve both privacy and utility if the adversary is too powerful, hence we propose here to study adversaries with restricted power, based on placing a bound on his prior knowledge. We will show that for certain values of this bound privacy and utility is possible.

Specifically, we restrict the adversary by assuming that for every tuple t his prior probability $Pr_1[t]$ is either small, or equal to 1. In the first case we will seek to hide the tuple from the adversary; in the second case there's nothing to do, since the adversary already knows t. Another way to look at this is that we require the algorithm to preserve the privacy for tuples for which $Pr_1[t]$ is small: if $Pr_1[t]$ is large, we may well assume that it is 1, in essence giving up any attempt to hide t from the adversary.

Definition 2.1. *Let $d \in (0, 1)$. A d-bounded adversary is one for which $\forall t \in D$, either $Pr_1[t] \leq d$ or $Pr_1[t] = 1$.*

We also consider a sub-class of d-bounded adversaries called the d-independent adversary, which further require Pr_1 to be tuple-independent:

Definition 2.2. *A d-independent adversary is a d-bounded, tuple-independent adversary.*

Thus, for a d-independent adversary there can be no correlations among tuples. However within a tuple there may be correlations among the attributes. For example consider a table with the schema (`age`, `nationality`, `disease`). A d-independent adversary who knows the correlation "*Japanese* have a low probability of having *heart attack*" corresponds to the distribution in which every tuple having the nationality *Japanese* and the disease *heart attack* has a low prior probability.

2.2 Privacy definition and motivation

Our definition of privacy is based on comparing the prior probability $Pr_1[t]$ with the a posteriori probability $Pr_{12}[t \mid V]$, and is derived from existing definitions in the literature.

Definition 2.3. *An algorithm is called (d, γ)-private if the following holds for all d-independent adversaries Pr_1, views V, and tuples t s.t. $Pr_1[t] \leq d$:*

$$\frac{d}{\gamma} \leq \frac{Pr_{12}[t|V]}{Pr_1[t]} \quad \textit{and} \quad Pr_{12}[t|V] \leq \gamma$$

If tuple t fails the left (or right) inequality then we say there has been a negative (or positive) leakage, respectively.

Our definition of (d, γ)-privacy is an adaptation of the notion of a (ρ_1, ρ_2) breach defined in [9]. A positive (ρ_1, ρ_2) breach, for $\rho_1 < \rho_2$, is defined when the prior is $Pr[t] \leq \rho_1$ and the posteriori is $Pr[t|V] > \rho_2$. Thus a positive leakage (right inequality) in (d, γ)-privacy is precisely a (d, γ) breach ($\rho_1 = d, \rho_2 = \gamma$). On other hand, a negative (ρ_1, ρ_2) breach in [9] does not correspond to our setting, because it is meant to capture large values of the prior, while in our case the prior is $\leq d$. Instead, our definition for negative leakage is relative, i.e. the a posteriori cannot drop by more than a factor of $\rho_1/\rho_2 = d/\gamma$.

The privacy definition assumed in [9, 13, 11] is based on variations of the (ρ_1, ρ_2)-breach (called the *Uniformative principle* in [13]). In [8, 3] privacy is based on a different definition called ϵ-indistinguishability in [8] and γ-amplification in [3]: we show in our technical report [16] how those related to (d, γ)-privacy.

2.3 Utility

Our definition of utility is based on how well one can estimate *counting queries*, i.e. queries of the form *count the number of records satisfying a certain predicate*. Formally:

Definition 2.4. *A* counting query *is a query of the form:*

```
select count(*) from I  where C
```

where C is any condition on the attributes $A_1, A_2, \ldots, A_l$ of the database I. We denote with $Q(I)$ the result of evaluating Q on an instance I; thus, $Q(I)$ is a number.

We illustrate with two examples:

```
Q1 = select count(*)
     from I
     where (age > 16 and age < 17 or age > 55)
       and score > 91

Q2 = select count(*)
     from I
     where score > 98 and nationality in
             (select country from Regions
              where continent='Africa'
                                    and GDP < 10000)
```

The first query counts the number of either very young or very old people that scored > 91. The second query counts the number of testers with score over 98 and coming from a poor African country. This query uses an auxiliary table `Regions` to check the continent of a country and its GDP to determine if that country is poor. In general the `where` condition in a query Q may use arbitrarily complex predicates over one, or several attributes.

Our notion of utility is based on how well the users can approximate counting queries. Users do not have access to the database I, only to a view V. The anonymization algorithm must specify how users can estimate query answers by rewriting a query Q expressed over I into an *estimator* $\tilde{Q}$ expressed over V, s.t. $\tilde{Q}(V)$ approximates $Q(I)$. The estimator will examine the tuples in V, apply any necessary complex predicates, and perform other computations. The only information that it cannot access is the secret database, I. Our formal notion of utility requires that the absolute error $|Q(I) - \tilde{Q}(V)|$ be small, for all queries Q

Definition 2.5. *A randomized algorithm A is called (ρ, ϵ)-*useful *if it has an estimator s.t. for any query Q and instance I:*

$$Pr_2\Big[\ |Q(I) - \tilde{Q}(V)| \geq \rho\sqrt{n}\ \Big]\ \leq\ \epsilon$$

Thus $\rho\sqrt{n}$ represents a bound on the absolute error, guaranteed with a probability of at least $1 - \epsilon$.

2.4 Discussion

Choice of privacy parameters An intuitive reference point for the bound d of the adversary is n/m, since the database has n tuples from a domain of size m, so we should expect that the adversary's prior is $Pr_1[t] \geq n/m$ at least for some tuples t; in [2] privacy was also measured with respect to a parameter 's' equal to n/m. More precisely, the bound on the adversary will be $d = kn/m$, for some $k > 1$. The a posteriori, γ, is larger, and measured in absolute values. For example, $\gamma = 0.05$ means that the view V leaks no tuple t with more than 5% probability.

Choice of utility parameters We discuss now our particular choice for the definition of utility (Def. 2.5). A first observation is that if the algorithm could guarantee $|Q(I) - \tilde{Q}(V)| = O(1)$ then privacy is compromised: the attacker simply counts the number tuples in the databases that are equal to a given tuple, say $\texttt{Age} = 28$, $\texttt{Nationality} = \texttt{Indian}$, $\texttt{Score} = 99$. The real answer can be only 0 or 1, depending on whether the tuple is present in I or not, hence, if the estimator is too accurate then there is a privacy leakage. Thus, the absolute error cannot be $O(1)$. Dinur and Nissim [6] showed that, in a different setting from ours, if queries can be estimated with an absolute error of $o(\sqrt{n})$, then there would be massive privacy breaches. That result requires the estimator to guarantee the error deterministically. Nevertheless, we can adapt their results to our setting and prove the following in our technical report [16]:

Theorem 2.6. *Let A be any randomized algorithm. If there exists $\rho = o(1)$ s.t. for all I and for all queries Q, we have $Pr_2(|\tilde{Q}(V) - Q(I)| \leq \rho\sqrt{n}) > 1/2$, then there exists a tuple t such that $Pr_1[t]$ is small ($\leq n/m$) but $Pr_1[t|V]$ is large ($\geq 2/3$).*

This explains that the absolute error cannot be $o(\sqrt{n})$ either, and justifies our choice of expressing the absolute error as $\rho\sqrt{n}$. In practice, one wants ρ to be some small constant.

k-Anonymity We argued in the introduction that the k-anonymity algorithm do not satisfy our formal notions of privacy and utility; we justify this here briefly, referring to the k-anonymity and l-diversity variant, introduced in [13]. Considering a d-independent adversary, privacy is compromised when the adversary knows that some tuples have very *low* probability. To see the intuition, consider the instance in Table 1(b). Suppose a d-independent adversary is trying to find out Joe's test score, and he knows that Joe is likely to have a low score: i.e. $Pr_1[t]$ is very low for a tuple saying that Joe's score is greater than 95, and larger (but still $\leq d$) for tuples saying that his score less than 90. If the adversary knows that Joe's age is less than 30, then his record is among the first three: since the first two tuples have scores 99 and 97, they have very *low* probability. The adversary, thus, concludes that Joe's scores is 82 with very *high* probability. Notion of (ρ, ϵ)-usefulness is also not satisfied. Suppose we want to estimate the number of students between 29 and 31 years old from Table 1(c): the answer can be anywhere between 0 and 6, and, if our estimate is the average, 3, then the only way we can guarantee any accuracy is by making assumptions on the distribution of the data.

3. IMPOSSIBILITY RESULT

Our main impossibility result says that if the attacker is powerful (i.e. his prior is large, $d = \Omega(n/\sqrt{m})$), then no algorithm can achieve both privacy and utility. This holds even if the attacker's prior is tuple-independent. For the impossibility result, we first show that if $d = \Omega(n/\sqrt{m})$ then no (d, γ)-private algorithm can provide even a weak notion of utility, called meaningfulness, which was first considered in [8]. We then prove that any algorithm that has some utility must be meaningful. Meaningfulness is based on the notion of statistical difference:

Definition 3.1. *The statistical difference between two distributions Pr_A and Pr_B over the domain X is $SD(Pr_A, Pr_B)$ $= \sum_{x \in X} \mid Pr_A(x) - Pr_B(x) \mid$.*

Note that $SD(Pr_A, Pr_B) \in [0, 2]$: it is 0 when $Pr_A = Pr_B$, and is 2 when $\forall x.(Pr_A(x) = 0 \vee Pr_B(x) = 0)$. We explain now informally the connection between statistical difference and utility. Suppose an algorithm A gives reasonable estimates to counting queries Q. Let Q be a "large" query, i.e. if executed on the entire domain it returns sizeable fraction, say 1/5. When we estimate Q on the view published for a particular instance I we will get some errors, which depend on actual size $n = |I|$. If there is any utility to A then a user should be able distinguish between the two extreme cases, when $Q(I) = n$ and when $Q(I) = 0$. To capture this intuition, define the uniform distribution $Pr_1(I)$ as follows: if $|I| = n$, then $Pr_1[I] = \frac{1}{\binom{m}{n}}$, and if $|I| \neq n$, then $Pr_1[I] = 0$. Thus, Pr_1 makes every instance of size n equally likely. Let E_Q be the event $(|I| = n \wedge Q(I) = n)$, and E'_Q the event $(|I| = n \wedge Q(I) = 0)$. Then we expect to be able to differentiate between the following two distributions: $Pr^Q_A(V) = Pr_{12}[V|E_Q]$ and $Pr^Q_B(V) = Pr_{12}[V|E'_Q]$. To obtain a reasonable estimate for Q, $SD(Pr^Q_A, Pr^Q_B)$ should be large. On the other hand, if $SD(Pr^Q_A, Pr^Q_B)$ is very small then no reasonable estimate of the query can be obtained from any of the published views. An algorithm is meaningless if the $SD(Pr^Q_A, Pr^Q_B)$ is small for a large fraction of the queries Q. An algorithm is meaningful if it is not meaningless.

Definition 3.2. *Let $f < 1$ be a constant independent of the domain size m and database size n. Let Q be a counting query and $Q(D)$ represent the number of tuples in the domain D which satisfy Q. Consider all queries Q s.t $\frac{1}{2}(1 - f) \leq \frac{Q(D)}{m} \leq \frac{1}{2}(1+f)$. An algorithm A is called meaningless if $SD(Pr^Q_A, Pr^Q_B)$ is smaller than $1/2$ for a fraction $2/3$ of queries Q.*

Next we state our impossibility result using meaningfulness as the definition of utility. For the sake of concreteness we use the constant 1/2 for statistical difference and 2/3 for the fraction of queries in the definition of meaningfulness. However, the impossibility result works for arbitrary constants.

Theorem 3.3. *Let A be a meaningful algorithm. Then for all $\gamma < 1$ there exists a constant c, independent of m,n and γ such that A is not (d, γ)-private for any $d \geq \frac{1}{c}\frac{\gamma}{1-\gamma}\frac{n}{\sqrt{m}}$. In particular no meaningful algorithm can offer $(k\frac{n}{m}, \gamma)$-privacy for $k = \Omega(\sqrt{m})$*

Intuitively, the theorem states that if we fix γ as the bound on the posterior then any algorithm providing meaningful utility cannot provide (d, γ)-privacy for $d = \Omega(n/\sqrt{m})$. Absence of (d, γ) privacy means that there is a d-independent adversary for which the following happens for some tuple t. There is a positive leakage ($Pr_1[t] \leq d$ but $Pr_{12}[t|V] \geq \gamma$) or there is negative leakage ($Pr_{12}[t|V]$ is much smaller than $Pr_1[t]$)

To make the theorem 3.3 strong, we used meaningfulness which is a very weak notion of utility. Our notion of (ρ, ε)-usefulness is much stronger as shown by the proposition below. Thus, the result in theorem 3.3 applies to (ρ, ε)-usefulness as well.

Proposition 3.4. *Let A be any (ρ, ε)-useful algorithm. If $\rho = o(\sqrt{n})$ then for any query Q, $SD(Pr^Q_A, Pr^Q_B) > 2(1-2\varepsilon)$. In other words, any useful algorithm is meaningful.*

4. ALGORITHM

4.1 The $\alpha\beta$ Algorithm

The $\alpha\beta$ algorithm (Algorithm 1) takes as input a database instance I and publishes a view V. We assume that I has l attributes A_1,A_2,...,A_l. Each attribute A_i takes values from a finite domain D_i. We denote $D = D_1 \times \ldots \times D_l$ the domain of all tuples with attributes $A_1, \ldots, A_l$, thus, $I \subseteq D$, and we write $m = |D|$.

Algorithm 1 $\alpha\beta$ (**Inputs:** $I, \alpha, \beta, D_1, .., D_l$)

```
Let $D = D_1 \times D_2 \times \ldots \times D_l$
**α-step** For every tuple in $I$, insert it in the view $V$ independently with probability $\alpha + \beta$.
**β-step** For every tuple in $D$ which is not in $I$, insert it in the view $V$ independently with probability $\beta$. (Algorithm 2 shows how to do this efficiently)
Publish $V$, $\alpha$, $\beta$ and $D_1$,..,$D_l$
```

The large size of D makes a naive implementation of the $\alpha\beta$ algorithm impossible. Specifically the β-step, which requires going through all tuples in the domain and inserting them with probability β, will have time complexity $O(m)$. However, note that the expected number of tuples to be inserted is βm, which is much smaller. We shall see in Sec. 4.2 that $\beta m \leq (k/\gamma)n$. Algorithm 2 is a simple randomized algorithm which runs in expected time of $O(\beta m)$. It simulates, exactly, the β-step of the $\alpha\beta$ algorithm.

Algorithm 2 β-step **Inputs:** β, $D_1, .., D_l$

```
1: Pick a random number $r$ according to the binomial distribution with parameters $m - n$ and $\beta$. The probability that $r$ equals $i$ is thus $\binom{m-n}{i}\beta^i(1-\beta)^{m-n-i}$
2: Insert $r$ tuples into the view as follows:
3: **repeat**
4:    For all attributes $A_j$, pick a value $a_j \in D_j$ uniformly at random
5:    Let $t = (a_1, a_2, \ldots, a_l)$
6:    **if** $t$ not in $V \cup I$ **then**
7:       insert $t$ in $V$
8:    **end if**
9: **until** $r$ tuples have been inserted in $V$
```

Note that in step 4 of the β-step algorithm, we may pick a tuple which is already in the view. This would result in an unnecessary increase in the run time of the algorithm. However, the probability of a tuple being picked again is very low as $\beta m \ll m$. A nice property of the β-step algorithm is that if T denotes the actual run time of the algorithm and T_0 the expected run time, then the probability that T is greater than T_0 by a additive factor of c, goes down exponentially with c.

Next we give the estimation algorithm. Given a view V the goal is to obtain an estimate $\tilde{Q}(V)$ which approximates the actual answer $Q(I)$. Recall that α and β are published

Algorithm 3 Estimation **Inputs:** V,Q,α,β,$D_1, .., D_l$

```
Let $D = D_1 \times \ldots \times D_l$.
Compute $n_V = Q(V)$; that is $n_V$ is the result of the query evaluated on the view $V$.
Compute $n_D = Q(D)$; that is $n_D$ is the query evaluated on the entire domain (we explain below how to do this efficiently)
$\tilde{Q}(V) = \frac{n_V - \beta n_D}{\alpha}$
```

Example 4.1 We illustrate the estimation algorithm using the Table 1(a) as an example. Table 1(d) is a sample view published by the $\alpha\beta$ algorithm. Domain for the attributes *age*, *nationality* and *score* are the sets [20,39], {American, British, Indian} and [81,100] respectively. $m = 20 \times 3 \times 20 = 1200$ and $n = 6$. The parameters used are $\alpha = 2/3$ and $\beta = 2\alpha n/m = 1/150$. Suppose a researcher has the hypothesis '*Test score of an individual is less than three times his age*'. For this purpose the following query Q needs to be evaluated:

```
Q = select count(*)
    from I
    where score < 3*age
```

n_D, the number of tuples in the entire domain which satisfy Q, is 190. $n_V = 6$ is the query evaluated on the view. The estimate, $\tilde{Q}(V)$, is $\frac{n_V - \beta n_D}{\alpha} = \frac{6-190/150}{0.66} = 7.1$ while the original answer $n_0 = 4$.

Computing Q(D) In the rest of this subsection we discuss how to compute Q on the domain D, line 3 of the Algorithm 3.

All prior methods (like privacy algorithms for OLAP) assume that the `where` condition in the query is a conjunction of simple predicates, i.e. each predicate involves one attribute. An example of such a query is (`score` $<$ 90)`and`(`age` $>$ 30). But the predicate `score` $< 3 *$ `age` in Example 4.1 is not of this form. We explain here how to compute $Q(D)$ for queries with arbitrarily complex conditions, C. Our discussion is informal, and we refer to our technical report [16] for the formal algorithm.

For any boolean condition C, denote $p(C)$ the probability that a randomly chosen tuple $t \in D$ satisfies C: $p(C) = |\{t | t \in D, C(t)\}|/m$ (recall: $m = |D|$). It suffices to show how to compute $p(C)$, because $Q(D) = p(C)m$.

We express C in DNF, thus, $C = C_1 \vee C_2 \vee \ldots \vee C_k$, and apply inclusion/exclusion to $p(C) = \sum_i p(C_i) - \sum_{i,j} p(C_i, C_j) + \ldots$ Thus, it suffices to show how to compute $p(C)$ where C is a conjunction of atomic predicates. For that, we rely on three ideas.

First, if C is an atomic predicate on, say, two attributes A_i, A_j then we evaluate $p(C)$ by iterating over $D_i \times D_j$. In the example above, C is `score` $< 3 *$ `age`, and we can evaluate $p(C)$ by brute force, by iterating over the domain of all (`score`, `age`) pairs. Denoting $m_k = |D_k|$, for $k = 1, \ldots, l$, the cost of computing $p(C)$ over attributes A_i, A_j is $m_i m_j$: note that this is much smaller than $m = m_1 m_2 \ldots m_l$.

Second, if $C = C_1 \wedge C_2$ and C_1, C_2 are conditions that refer to distinct sets of attributes, then $p(C) = p(C_1)p(C_2)$. This is obvious, since C_1, C_2 are independent.

Third, suppose $C = C_1 \wedge C_2$ and the predicates in C_1 and C_2 share some common attributes, say A_i and A_j. Then we consider all pairs of constants $a \in D_i$ and $b \in D_j$. We create $m_i m_j$ predicates $C_1[a/A_i, b/A_j]$ and similarly $C_2[a/A_i, b/A_j]$, and compute their probabilities. Here a/A_i means that we substitute all occurrences of the attribute A_i with the constant a, and similarly for b/A_j. Finally, $C_1[a/A_i, b/A_j]$ and $C_2[a/A_i, b/A_j]$ are independent, hence $p(C) = \sum_{a \in D_i, b \in D_j} p(C_1[a/A_i, b/A_j]) p(C_2[a/A_i, b/A_j])$. The cost is a multiplicative factor $m_i m_j$.

Thus, the algorithm's running time depends on the largest number of attributes shared between two expressions C_1, C_2. This depends on how we factor the conjunction C, and is related to the tree width [21].

Example 4.2 Consider the condition: $C = P_1(A_1, A_2) \wedge P_2(A_2, A_3) \wedge P_3(A_3, A_4)$, where P_1, P_2, P_3 are three complex predicates (e.g. $P_1 = A_1^2 + A_2^2 \geq 20$). We write C as follows:

$$\begin{aligned} C &= C_1 \wedge P_3(A_3, A_4) \\ C_1 &= P_1(A_1, A_2) \wedge P_2(A_2, A_3) \end{aligned}$$

We proceed top-down. To evaluate C we note that the two conjuncts share the common attribute A_3, so we substitute A_3 with every constant $a_3 \in D_3$, and compute the probabilities of $P_3(a_3, A_4)$, and of:

$$C_1[a_3/A_3] = P_1(A_1, A_2) \wedge P_2(A_2, a_3)$$

The probability of $P_3(a_3, A_4)$ is obtained by iterating over all $a_4 \in D_4$ and checking if the condition $P_3(a_3, a_4)$ is true. The probability of $C_1[a_3/A_3]$ is obtained recursively. Here the two conjuncts share A_2, so we iterate over all $a_2 \in D_2$ and compute the probabilities of $P_1(A_1, a_2)$ and $P_2(a_2, a_3)$. Our discussion leads to the following formula, where $p(P_1(a_1, a_2))$ is $\frac{1}{m}$ if (a_1, a_2) satisfies the predicate P_1 and is 0 otherwise (and similarly for P_2, P_3):

$$\begin{aligned} p(C) = & \sum_{a_3 \in D} (\sum_{a_2 \in D_2} \sum_{a_1 \in D_1} p(P_1(a_1, a_2)) p(P_2(a_2, a_3)) \\ & \sum_{a_4 \in D_4} p(P_3(a_3, a_4))) \end{aligned}$$

The time complexity is $m_3(m_1 m_2 + m_4)$.

4.2 Privacy and utility analysis

The following theorem shows that the $\alpha\beta$ algorithm satisfies (d, γ)-privacy.

Theorem 4.3. *The $\alpha\beta$ algorithm is (d, γ)-private where $d \leq \gamma$ if we choose α and β such that $\frac{\beta}{\alpha+\beta} \geq \frac{d(1-\gamma)}{\gamma(1-d)}$ and $\alpha + \beta \leq 1 - \frac{d}{\gamma}$*

As shown in our technical report [16], any (d, γ)-private algorithm satisfies differential privacy, the notion of privacy described in [7]. This means, in particular, the $\alpha\beta$-algorithm satisfies differential privacy as well.

Next we show that the $\alpha\beta$-algorithm satisfies the definition of (ρ, ϵ)-usefulness.

Theorem 4.4. *Let r be a constant and α, β be such that $\beta \leq r\alpha^2 \frac{n}{m}$. Then, the $\alpha\beta$ algorithm is (ρ, ϵ)-useful for $\rho = \sqrt{2(r+1)ln(\frac{2}{\epsilon})n}$*

Choice of parameters We wish to ensure $(\frac{kn}{m}, \gamma)$-privacy, where k is a constant, i.e. $d = \frac{kn}{m}$ is the bound on the adversary's prior probability, and γ is the bound on the posteriori probability that is acceptable for us. We also want to ensure utility. Choosing $\alpha + \beta = \frac{1}{2}$ satisfies both Theorems 4.3 (assuming $d/\gamma < 1/2$) and 4.4. Next, we choose $\beta = \frac{k}{\gamma}\frac{n}{m} = \frac{d}{\gamma}$. This satisfies Theorem 4.3 (since $1 > \frac{1-\gamma}{1-d}(\alpha + \beta)$), and also satisfies Theorem 4.4 whenever $\frac{k}{\gamma} \leq \frac{r}{4}$.

In other words, with these choices of parameters we have $(k\frac{n}{m}, \gamma)$-privacy and (ρ, ϵ)-usefulness for $\rho = 2\sqrt{2\frac{k(1-\gamma)}{\gamma}ln(\frac{2}{\epsilon})}$. The privacy/utility tradeoff is like this. We want to protect against a powerful adversary, $k \gg 1$, and want to guarantee a small a posteriori, $\gamma \ll 1$; and this limits the bound on the estimate's error as $\rho \approx \sqrt{k/\gamma}$.

Comparison with randomized response algorithms The FRAPP method has the best tradeoff among the local perturbation algorithms. The tradeoff for that method can be expressed as $\rho = k/\gamma$. On the other hand the $\alpha\beta$ algorithm has a much better tradeoff of $\rho = \sqrt{k/\gamma}$. Thus for the same privacy (i.e k and γ) the $\alpha\beta$ algorithm yields smaller errors as compared to other methods. This becomes especially important since the error bound is actually $\rho\sqrt{n}$ which means that even a small change in ρ gets amplified, as $\sqrt{n}$ is often large. We support our claim by providing empirical results as described in the case study.

4.3 Comparison to FRAPP

FRAPP [3] provides a general framework for representing randomized response algorithm. Many randomized operators like 'select a size' [9] and MASK [18] were described using this framework. We describe here their framework, then compare it to the $\alpha\beta$ algorithm.

The database instance and the published view are denoted by column vectors $\mathbf{x}$ and $\mathbf{y}$, respectively, of size m. The randomized response algorithm is represented as a $m \times m$ matrix A whose entry a_{pq} represents the probability that the tuple p of the database gives rise to the tuple q in the view. Given the matrix A and the input database $\mathbf{x}$, the view $\mathbf{y}$ is generated by a localized random process. The random process, in some sense, can be succinctly represented as $\mathbf{y} = A\mathbf{x}$.

[3] show that the utility of the algorithm increases when the *condition number* of the matrix A decreases: the condition number is the ratio between the largest and the smallest eigenvalue. On the other hand, the privacy constraint requires that the ratio of any two entries in the same row of the matrix remains bounded. Under this condition [3] prove that the matrix that achieves optimal privacy/utility tradeoff is as follows: diagonal elements are γ_{frapp}, and all non-diagonal elements are $\frac{1-\gamma_{frapp}}{m-1}$.

Our $\alpha\beta$ algorithm strictly generalizes the FRAPP framework by taking advantage of the fact that it can use a central server. Formally, the transformation in our algorithm is described as $\mathbf{y} = A\mathbf{x} + \mathbf{b}$. Thus the random process effectively adds a noise vector $\mathbf{b}$ to $A\mathbf{x}$. Intuitively speaking, this provides privacy as the effective ratio of entries in a row remains bounded. Moreover, utility improves because of the lower condition number of A.

For the $\alpha\beta$ algorithm, the matrix A is the diagonal matrix with each diagonal entry equal to α. The noise vector $\mathbf{b}$ is the uniform vector with all entries equal to β.

4.4 Extensions

4.4.1 Updates

Suppose the data owner updates the instance I, and thus obtains several versions $I_1, I_2, \ldots$ If the data owner runs the $\alpha\beta$ algorithm separately on each version, she obtains a sequence of views $V_1, V_2, \ldots$ This, however, creates a severe privacy breach. For example, consider the case when a single tuple t occurs in each of the published views. As β is significantly smaller than α, that the probability that t was inserted in each of the views, when it does not belong to the original database, is very small. This results in an obvious privacy breach with respect to t. The problem is that the instances $I_1, I_2, \ldots$ are highly correlated, i.e. two consecutive instances share a large number of tuples. To avoid this scenario, we describe Algorithm 4 below. It allows us to publish multiple views while providing the same privacy. For readability we consider only the extreme case, when I remains unchanged (this is the worst case), i.e. $I_1 = I_2 = \ldots$ The algorithm needs a parameter v, which represents a bound on the number of views published.

Algorithm 4 Publishing multiple views

```
Let $V_1$, $V_2$, ..., $V_{i-1}$ be the views published so far.
Compute $V' = \cup_{j=1}^{i-1} V_j$.
Generate the view $V_i$ as follows:
start
    For each tuple in the original instance $I$, insert it in $V_i$ with probability $\alpha + \beta$
    For each tuple in $V' - I$, insert it in $V_i$ with probability $(\alpha + \beta)e^{-\frac{1}{\mathsf{v}}}$
    For every other tuple in the domain, insert it in $V_i$ with probability $[1 - (\alpha + \beta)]^{i-1}\beta$
end start
Publish $V_i$
```

We discuss three aspects of publishing multiple views: we analyze the privacy, we examine the total size (note that we insert more tuples for each new version), and we examine its utility.

For privacy, note that once a noisy tuple has been inserted in some published view, the algorithm increases the probability of its insertion for future views. This makes it hard to determine whether a tuple occurs in the database even if it occurs in a lot of published views, and allows us to prove in the technical report [16] that the algorithm is (d, γ)-private.

The size of the views increases, but it is bounded. To see that note that the probability of inserting a new noisy tuple, i.e. a tuple outside V', becomes exponentially smaller with each published view. It can be shown that the size of any view is not more than the size of V_1 (the first published view) times $\frac{1}{\alpha+\beta}$. This gives an upper bound on the size of all published views, which is independent of the number of views.

The utility of a view V_i in the sequence is slightly less than that for the first view. The estimator for a query Q on the view V_i is

$$\tilde{Q}(V_i) = \frac{n_{V_i} - (\alpha + \beta)e^{-\frac{1}{\mathsf{v}}} n_{V'} - [1 - (\alpha + \beta)]^{i-1}\beta n_d}{(\alpha + \beta)(1 - e^{-\frac{1}{\mathsf{v}}})}$$

Here n_{V_i} is the answer of the query Q evaluated over the view V_i; $n_{V'}$ is the answer of the query Q evaluated over the union V' of all the previously published views; n_d is the answer evaluated over the entire domain. It can be shown that by using this estimator, the privacy utility tradeoff of the view V_i can be expressed as $\rho = \mathsf{v}\sqrt{\frac{k}{\gamma}}$

4.4.2 Non-uniform Noise Vector

So far we considered arbitrary d-independent adversaries, i.e. tuple independent adversaries with the only restriction that tuple probability $< d$. The prior for such adversaries can be very different from the underlying distribution from which the database was drawn. For example consider a database with schema (age, nationality, disease). *Flu* is a more common disease than *cancer*. Thus, according to the source distribution, the probability of a person having flu is higher than her having cancer. An adversary, however, might know that the person X has a fatal disease. For that particular X, the prior would have a higher probability for cancer.

However, if we know the attacker's prior then we can improve the privacy/utility tradeoff by generating the random noise according to this prior. More precisely, we modify the noise vector $\mathbf{b}$ according to a source distribution, the distribution which models the prior. Adding noise according to the source distribution allows us to add the smallest amount of noise, while providing the same privacy. Adding a smaller amount of noise enables better utility. We modify the $\alpha\beta$ algorithm as follows. Instead of inserting tuples with uniform probability β, we do so with probability $\beta(t)$, a function proportional to $source(t)$. We consider the following source distribution: Each value a in the domain of attribute A_i has an associated source probability, $p(a)$; the source distribution is attribute-independent, i.e. $source(t) = p(t.A_1) \cdots p(t.A_l)$.

This demands changes in both the β-step of the $\alpha\beta$ algorithm as well as the estimation algorithm. The β-step can be approximated as:

Algorithm 5 Generalized β-step

```
Compute the binomial distribution corresponding to parameters $m - n$ and $\frac{\sum_{t \in (D-I)} source(t)}{m-n}$
Generate a random number $r$ using this binomial distribution
Insert $r$ tuples into the view using the following procedure:
repeat
  For all attributes $A_j$, pick a value $a_j \in D_j$ randomly according to the distribution $p(a_j)$
  Let $t = (a_1, a_2, \ldots, a_l)$
  if $t$ not in $I$ then
    $V = V \cup t$
  end if
until $r$ tuples have been inserted in $V$
```

The estimate for query Q also changes. For a fixed β, it was simply $\frac{n_v - \beta n_d}{\alpha}$, where n_D denotes the number of tuples in the domain which satisfy the query, i.e. $n_D = |Q(D)|$. Now, instead of βn_D, we need to evaluate $\sum_{t \in (Q(I))} \beta(t)$.

5. EXPERIMENTS

To compare the privacy utility trade off of the $\alpha\beta$ algorithm and the FRAPP method, we experimented with US census data in the Adults database. The database is widely used in experiments for illustration of privacy preserving algorithms [13, 4]. The database I has 9 attributes and $n = 30162$ tuples. The product of the attributes active domains has $m = 648023040$ tuples, thus $n/m \approx 4.65 \times 10^{-5}$. For both methods, the parameters were chosen so as to ensure $(10n/m, 0.2)$-privacy: i.e. the adversary's prior is bounded by $d = 0.0465\%$ and the posterior by $\gamma = 20\%$. Privacy for both the methods was, thus, fixed to the same value.

We compared the utility of both the methods, by running all possible selection queries with up to three attributes. That is, for every subset of (up to) three attributes A_i, A_j, A_k, $i, j, k \in \{1, \ldots, 9\}$, and every three values $a \in D_i$, $b \in D_j$, $c \in D_k$, we evaluated the query:

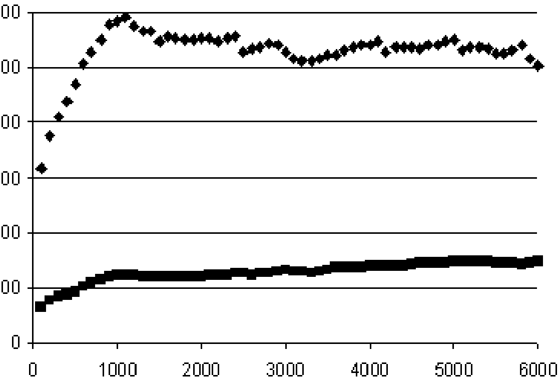


Figure 3: Graph of average cumulative error vs $Q(I)$ for the FRAPP and $\alpha\beta$ algorithm: Upper curve corresponds to the FRAPP algorithm showing around 4.5 times larger errors on average than the $\alpha\beta$ algorithm

```
select count(*)
from I
where Ai = 'a' and Aj = 'b' and Ak = 'c'
```

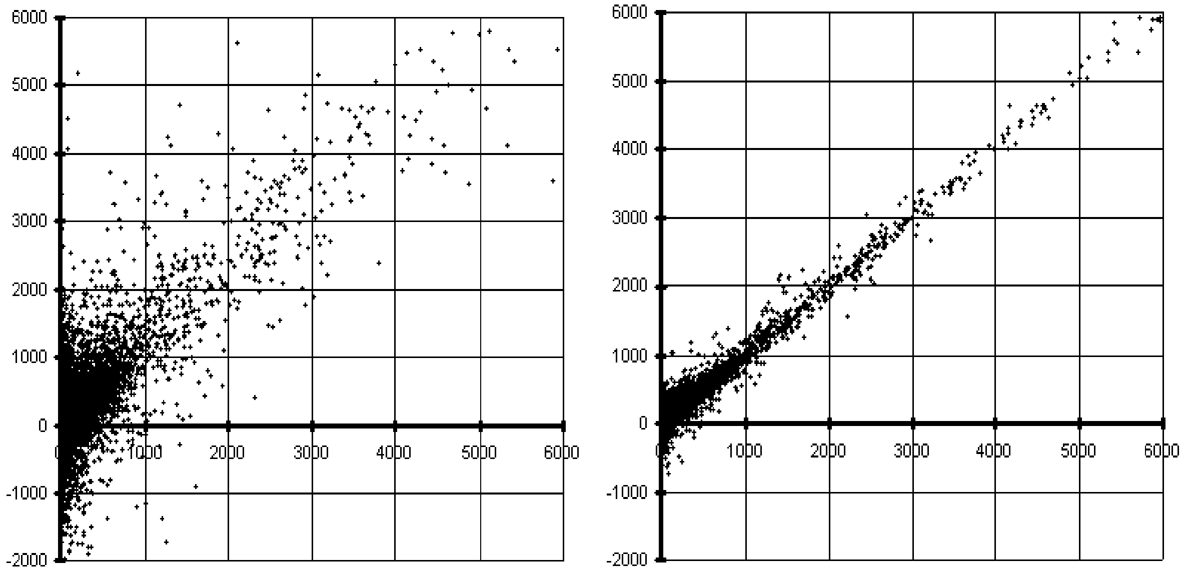
Figure 2 shows the result of comparison. Each dot represents one query Q, where $x = Q(I)$ and $y = \tilde{Q}(V)$: thus, a perfect estimator would correspond to the $y = x$ line. It is evident that the $\alpha\beta$ algorithm provides much better estimates and hence has better utility. Figure 2 points to another interesting property: For small values of $Q(I)$ the estimated values are far off (even negative); this is necessary to preserve privacy. But for large values of $Q(I)$ the estimations are much better and meaningful.

As described earlier the tradeoff for $\alpha\beta$ algorithm is $\rho_1 \approx \sqrt{k/\gamma}$. Actually, the exact formula is $\rho_1 = 2\sqrt{2\frac{k(1-\gamma)}{\gamma} ln(\frac{2}{\epsilon})}$. While for FRAPP it is $\rho_2 = 2\frac{k(1-\gamma)}{\gamma}\sqrt{ln(\frac{2}{\epsilon})}$. Thus $\frac{\rho_1}{\rho_2} = \sqrt{\frac{2\gamma}{k(1-\gamma)}}$, which for our choice of k and γ is $\frac{1}{\sqrt{20}} \sim \frac{1}{4.5}$. However ρ_1 and ρ_2 are theoretical upper bounds on the error.

Figure 3 compares the observed values of ρ_1 and ρ_2. It is a graph of cumulative average error vs. $Q(I)$. There are two curves. Each point (x, y) on the curves has the following connotation: y is the average error $|\tilde{Q}(V) - Q(I)|$ taken over all queries with $Q(I) \geq x$. The upper curve represents the cumulative average errors for the FRAPP method while the lower curve corresponds to the $\alpha\beta$ algorithm. From the graph one can compute the approximate ratio of the observed values $\frac{\rho_1^{obs}}{\rho_2^{obs}}$. As evident from the graph it is $\sim \frac{120}{520} = \frac{1}{4.3}$ very close to the theoretically predicted value.

6. TUPLE CORRELATIONS

So far our analysis was restricted to tuple-independent adversaries. This makes the impossibility result very strong, but restricts the applicability of the $\alpha\beta$ algorithm. Here we discuss several extensions to tuple correlations. First, we



2(a): FRAPP algorithm [3]

2(b): $\alpha\beta$ algorithm

Figure 2: Graph of $\tilde{Q}(V)$ vs $Q(I)$ for the FRAPP and $\alpha\beta$ algorithm: Each query corresponds to a point. Points near the line $y = x$ have $\tilde{Q}(V) \sim Q(I)$ which means that query gets estimated well. Points lie on a band with the width of band signifying the extent of errors.

show that if the adversary knows arbitrary correlations, then no algorithm can achieve both privacy and utility. Then we examine a restricted class of correlations called exclusions, which is sufficiently general to model join/link attacks. For explaining the behavior of our algorithm with respect to these adversaries we need to distinguish between positive and negative leakage. Positive leakage happens when the adversary is able to determine with high probability that a specific tuple exists in the database. On the other hand, negative leakage occurs when it is possible to determine with high confidence that a specific tuple does not exist in the database. As described earlier, (d, γ)-privacy ensures absence of positive as well as negative leakage against all d-independent adversaries.

For adversaries with correlations restricted to exclusions, we show that our algorithm still guarantees privacy against *positive* leakages(utility, of course, is unchanged), but it cannot protect against negative leakage. The policy of protecting against positive leakage while permitting negative leakage is sometimes acceptable in practice, and it raises the question whether our algorithm could be strengthened in this case: after all the impossibility result in Theorem 3.3 is based on privacy definition for protecting against both negative and positive leakages. We answer this negatively, by giving a variant of the impossibility result for positive leakages only.

6.1 Arbitrary Correlations

Suppose we publish a view V of a database of diseases with schema (`age`, `zip`, `disease`) using our algorithm. An attacker knows the age and zip code for both Joe and Jim. Now Joe and Jim are brothers: if one has diabetes, then the other is quite likely to have diabetes as well (at least that's what the attacker believes). Suppose the attacker finds two tuples in V matching both Joe and Jim having diabetes. The probability that none was in I and were inserted by the algorithm is very small: β^2. In contrast, because of their strong correlation, the probability that they were in the instance I is now much larger: $Pr_1[t \cap t'] \gg Pr_1[t]Pr_1[t']$. Thus, upon seeing both tuples in the view the attacker concludes that they occurred in the original database with high probability. This is the reason why our algorithm has difficulties hiding data when the prior has correlations.

To the best of our knowledge, [14] is the only approach in literature that handles such correlations. However, the approach is designed for the case when there is a single sensitive attribute. It is not clear what the privacy utility tradeoff for the method would be, in presence of multiple sensitive attributes or when the distinction between sensitive and non-sensitive attributes is not made. Perhaps, the difficulty in dealing with such correlations is unavoidable. The following theorem shows that if the adversary is allowed arbitrary tuple correlations then any algorithm providing meaningful utility will have a positive leakage on some tuple. Theorem 6.1 is proved using a reduction from the negative result in [7]. The proof appears in our technical report [16].

Theorem 6.1. *Let A be any meaningful algorithm. Then for every $d = \Omega(n/m)$, there exists a d-bounded adversary for which $Pr_1[t] \leq d$ but $Pr_{12}[t|V] \geq 3/4$ for some tuple t.*

6.2 Exclusions

We consider now some restricted forms of correlations: exactly one tuple from a set of tuples occurs in the database.

Definition 6.2. *A d-exclusive adversary is a d-bounded adversary with the following kind of correlations among the tuples: There is a partition of D into a family of disjoint sets, $D = \bigcup_j S_j$, s.t.*

- *Tuples are pairwise independent if they do not belong to the same set.*
- *Exactly one tuple in each set occurs in the database instance*

This type of correlations models adversaries performing join/link attacks. Such attackers are able to determine some of the identifying attributes of a particular tuple, say the age and nationality of a person that they know must occur in the database. The adversary can thus identify a set of tuples S_j of the domain such that exactly one tuple in the set belongs to the database.

6.2.1 Positive results for d-exclusive adversaries

We have already seen the privacy analysis for the $\alpha\beta$ algorithm for (d, γ)-privacy, which is a privacy definition for protection against d-independent adversaries. In this section, we show that the algorithm provides a slightly weaker form of guarantee for d-exclusive adversaries. We show that the algorithm ensures that there is no positive leakage for d-exclusive adversaries: That is if $Pr_1[t] \leq d$ then $Pr_{12}[t|V] \leq \gamma$.

Theorem 6.3. *If $\alpha + \beta = \frac{1}{2}$ and $\beta \geq 2\frac{d}{\gamma}(\frac{1-\gamma}{1-d})$ then the algorithm ensures absence of positive leakage for all tuples and for all d-exclusive adversaries, i.e if $Pr_1[t] \leq d$ then $Pr_{12}[t|V] \leq \gamma$*

The algorithm cannot guarantee absence of negative leakage for d-exclusive adversaries, as seen from the following example:

Example 6.4 Suppose we publish a view V of a database of diseases with schema (`age`, `zip`, `disease`) using our algorithm. An attacker knows the age and zip code of a person Joe and additionally knows that no other person in the database has the same combination of age and zip code. The prior belief of the attacker is that Joe has exactly one disease but any one of the diseases is equally likely. Suppose the attacker finds a tuple t_1 in V matching Joe and Diabetes. Let us consider the tuple t_2 corresponding to Joe and Malaria. Theorem 6.3 shows that there won't be a positive leakage for t_1. However, for t_2 there is a drastic drop in the a posteriori probability causing a negative leakage.

6.2.2 Impossibility results for d-exclusive adversaries

In this section we extend the impossibility result for d-exclusive adversaries to accommodate the case when negative leakage is legally allowed and only positive leakage is considered as a privacy breach. The impossibility result shows that if an algorithm satisfies some form of weak utility then there exists a d-exclusive adversary, for $d = \Omega(\frac{n}{\sqrt{m}})$, for which there is positive leakage on some tuple.

The impossibility result holds only for a restricted class of randomized algorithms. However, the class is broad enough to encompass many of the privacy preserving algorithms in the literature. The class of algorithms which we consider satisfy the following bucketization assumptions:

- The algorithm is such that if the prior distribution is tuple independent then the posterior distribution is also almost tuple independent. More formally, the tuples in the domain can be partitioned into buckets such that the distribution $Pr_{12}[t_i|V]$ over the tuples obeys the following condition: If two tuples t_i and t_j lie in different buckets then they are independent conditioned on the view.
- Let the number of buckets be N_B. We assume that the distribution of tuples both of the database instance and the domain among the buckets is not too skewed. More formally , for every $k > 1$ there exists a $N_k < N_B$ such that if we remove any N_k buckets, the remaining still contain a fraction $1/k$ of tuples of I as well as D.

For example consider the method of full domain generalization (in [12]) as applied to ensure k-anonymity or the anatomy method (in [20]) as applied to ensure l-diversity. For both the methods the tuples corresponding to the different anonymized groups form buckets which satisfy the assumptions above.

The impossibility result for d-independent adversaries as well as the result for d-bounded adversaries had used meaningfulness as the utility definition. Impossibility result for d-exclusive adversaries requires a slight modification to this definition and is based on the notion of k-meaningfulness. Intuitively, k-meaningfulness works on the basis that a view, which publishes a small fraction of the database while not providing any information for other tuples, does not have good utility. More formally, a view V is k-meaningless if there is a small set S, where $S \subset I$, such that V does not provide any information for tuples outside S. We denote $|S|$ as $n(1-\frac{1}{k})$ where $1/k$ is a large fraction close to 1. The constraint that "V does not provide any information for tuples outside S" is represented as follows: Given a query Q, it is not possible to distinguish whether the answer to the query is $|Q \cap S|$ or $|Q \cap S| + \frac{n}{k}$, based on V. Thus, if E_Q is the event that $Q(I) = |Q \cap S|$ and E'_Q is the event $Q(I) = |Q \cap S|$, then the probability that V is published when E_Q occurs is approximately same as the probability when E'_Q occurs.

Definition 6.5. *Let S be a subset of I with $|S| \leq n(1 - \frac{1}{k})$. Let E_Q be the event that $Q(I) = |Q \cap S|$ and E'_Q be the event $Q(I) = |Q \cap S| + \frac{n}{k}$. Consider all queries Q s.t $\frac{1}{2}(1 - f) \leq \frac{Q(D)}{m} \leq \frac{1}{2}(1 + f)$. A view V is called k-meaningless if for a fraction 2/3 of queries Q, $\frac{1}{2} \leq \frac{Pr_{12}[V|E_Q]}{Pr_{12}[V]} \leq 2$ and $\frac{1}{2} \leq \frac{Pr_{12}[V|E'_Q]}{Pr_{12}[V]} \leq 2$*

A view is called k-meaningful if it is not k-meaningless. The following theorem shows that if the adversary is d-exclusive, then any algorithm, which satisfies the bucketization constraints and publishes k-meaningful views, will have positive leakage for some tuple.

Theorem 6.6. *Let A be any algorithm which satisfies the bucketization assumptions with $N_k \geq \frac{3}{d}$. Let V be any k-meaningful view published by A. Then there exists a constant c independent of n and m for which there is a d-exclusive adversaries with $d = max(\frac{3}{N_k}, c\frac{n}{\sqrt{m}})$ having positive leakage on some tuple.*

7. CONCLUSIONS

We have described a formal framework for studying both the privacy and the utility of an anonymization algorithm. We proved an almost tight bound between privacy and utility, based on the attacker's power. For the case where privacy/utility can be guaranteed, we have described a new, quite simple anonymization algorithm, based on random insertions and deletions of tuples in/from the database. We have done a limited empirical study, and saw a good privacy/utility tradeoff. An interesting problem for future work lies in bridging the gap between the impossibility result and the positive algorithm. In terms of d, the bound on adversary's prior, the gap can be expressed as between $O(n/m)$ and $\Omega(n/\sqrt{m})$.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. *SIGMODREC: ACM SIGMOD Record*, 29, 2000.

[2] R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving OLAP. In F. Özcan, editor, *SIGMOD Conference*, pages 251–262. ACM, 2005.

[3] S. Agrawal and J. R. Haritsa. A framework for high-accuracy privacy-preserving mining. Jan. 01 2005.

[4] E. K. C. Blake and C. J. Merz. UCI repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[5] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The suLQ framework. In *PODS: 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2005.

[6] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS: 22th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2003.

[7] C. Dwork. Differential privacy. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proceedings of ICALP (Part 2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Proceedings of Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[9] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS: 22th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2003.

[10] A. V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. *Inf. Syst*, 29(4):343–364, 2004.

[11] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *Proceedings of SIGMOD*, pages 217–228. ACM, 2006.

[12] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain K-anonymity. In F. Özcan, editor, *Proceedings of SIGMOD*, pages 49–60. ACM, 2005.

[13] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang, editors, *Proceedings of ICDE*, page 24. IEEE Computer Society, 2006.

[14] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *ICDE*, 2007.

[15] N. Mishra and M. Sandler. Privacy via pseudorandom sketches. In S. Vansummeren, editor, *PODS*, pages 143–152. ACM, 2006.

[16] V. Rastogi, D. Suciu, and S. Hong. The boundary between privacy and utility in data publishing, 2007. available from `www.cs.washington.edu/homes/vibhor/privacy.pdf`.

[17] S. P. Reiss. Practical data-swapping: The first steps. *ACM Transactions on Database Systems*, 9(1):20–37, Mar. 1984.

[18] S. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *VLDB*, pages 682–693. Morgan Kaufmann, 2002.

[19] L. Sweene. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[20] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim, editors, *Proceedings of VLDB*, pages 139–150. ACM, 2006.

[21] Y. Zabiyaka and A. Darwiche. Functional treewidth: Bounding complexity in the presence of functional dependencies. In A. Biere and C. P. Gomes, editors, *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 2006.