

# Slicing: A New Approach for Privacy Preserving Data Publishing

Tiancheng Li, Ninghui Li, *Senior Member, IEEE*, Jian Zhang, *Member, IEEE*, and Ian Molloy

**Abstract**—Several anonymization techniques, such as generalization and bucketization, have been designed for privacy preserving microdata publishing. Recent work has shown that **generalization loses considerable amount of information**, especially for **high-dimensional** data. Bucketization, on the other hand, does not prevent membership disclosure and does not apply for data that do not have a **clear separation** between quasi-identifying attributes and sensitive attributes. In this paper, we present a novel technique called slicing, which **partitions the data both horizontally and vertically**. We show that slicing preserves better data utility than generalization and can be used for membership disclosure protection. Another important advantage of slicing is that it can handle high-dimensional data. We show how slicing can be used for attribute disclosure protection and develop an efficient algorithm for computing the sliced data that obey the  $\ell$ -diversity requirement. Our workload experiments confirm that slicing preserves better utility than generalization and is more effective than bucketization in workloads involving the sensitive attribute. Our experiments also demonstrate that **slicing can be used to prevent membership disclosure**.

**Index Terms**—Privacy preservation, data anonymization, data publishing, data security.

## 1 INTRODUCTION

PRIVACY-PRESERVING publishing of microdata has been studied extensively in recent years. *Microdata* contains records each of which contains information about an individual entity, such as a person, a household, or an organization. Several microdata anonymization techniques have been proposed. The most popular ones are generalization [28], [30] for  $k$ -anonymity [30] and bucketization [34], [26], [17] for  $\ell$ -diversity [25]. In both approaches, attributes are partitioned into three categories: 1) some attributes are *identifiers* that can uniquely identify an individual, such as *Name* or *Social Security Number*; 2) some attributes are *Quasi Identifiers (QI)*, which the adversary may already know (possibly from other publicly available databases) and which, when taken together, can potentially identify an individual, e.g., *Birthdate*, *Sex*, and *Zipcode*; 3) some attributes are *Sensitive Attributes (SAs)*, which are unknown to the adversary and are considered sensitive, such as *Disease* and *Salary*.

In both generalization and bucketization, one first removes identifiers from the data and then partitions tuples into buckets. The two techniques differ in the next step. Generalization transforms the QI-values in each bucket into “less specific but semantically consistent” values so that tuples in the same bucket cannot be distinguished by their QI values. In bucketization, one separates the SAs from the QIs by randomly permuting the SA values in each bucket.

The anonymized data consist of a set of buckets with permuted sensitive attribute values.

### 1.1 Motivation of Slicing

It has been shown [1], [16], [34] that generalization for  $k$ -anonymity loses considerable amount of information, especially for high-dimensional data. This is due to the following three reasons. First, generalization for  $k$ -anonymity suffers from the curse of dimensionality. In order for generalization to be effective, records in the same bucket must be close to each other so that generalizing the records would not lose too much information. However, in high-dimensional data, most data points have similar distances with each other, forcing a great amount of generalization to satisfy  $k$ -anonymity even for relatively small  $k$ 's. Second, in order to perform data analysis or data mining tasks on the generalized table, the data analyst has to make the uniform distribution assumption that every value in a generalized interval/set is equally possible, as no other distribution assumption can be justified. This significantly reduces the data utility of the generalized data. Third, because each attribute is generalized separately, correlations between different attributes are lost. In order to study attribute correlations on the generalized table, the data analyst has to assume that every possible combination of attribute values is equally possible. This is an inherent problem of generalization that prevents effective analysis of attribute correlations.

While bucketization [34], [26], [17] has better data utility than generalization, it has several limitations. First, bucketization does not prevent membership disclosure [27]. Because bucketization publishes the QI values in their original forms, an adversary can find out whether an individual has a record in the published data or not. As shown in [30], 87 percent of the individuals in the United States can be uniquely identified using only three attributes (*Birthdate*, *Sex*, and *Zipcode*). A microdata (e.g., census data)

• T. Li, N. Li, and I. Molloy are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907.  
E-mail: {li83, ninghui, imolloy}@cs.purdue.edu.

• J. Zhang is with the Department of Statistics, Purdue University, West Lafayette, IN 47907. E-mail: jianzhan@purdue.edu.

Manuscript received 17 Feb. 2010; revised 26 Apr. 2010; accepted 19 July 2010; published online 18 Nov. 2010.

Recommended for acceptance by E. Ferrari.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-02-0097. Digital Object Identifier no. 10.1109/TKDE.2010.236.

usually contains many other attributes besides those three attributes. This means that the membership information of most individuals can be inferred from the bucketized table. Second, bucketization requires a clear separation between QIs and SAs. However, in many data sets, it is unclear which attributes are QIs and which are SAs. Third, by separating the sensitive attribute from the QI attributes, bucketization breaks the attribute correlations between the QIs and the SAs.

In this paper, we introduce a novel data anonymization technique called *slicing* to improve the current state of the art. Slicing partitions the data set both vertically and horizontally. Vertical partitioning is done by grouping attributes into columns based on the correlations among the attributes. Each column contains a subset of attributes that are highly correlated. Horizontal partitioning is done by grouping tuples into buckets. Finally, within each bucket, values in each column are randomly permuted (or sorted) to break the linking between different columns.

The basic idea of slicing is to break the association cross columns, but to preserve the association within each column. This reduces the dimensionality of the data and preserves better utility than generalization and bucketization. Slicing preserves utility because it groups highly correlated attributes together, and preserves the correlations between such attributes. Slicing protects privacy because it breaks the associations between uncorrelated attributes, which are infrequent and thus identifying. Note that when the data set contains QIs and one SA, bucketization has to break their correlation; slicing, on the other hand, can group some QI attributes with the SA, preserving attribute correlations with the sensitive attribute.

The key intuition that slicing provides privacy protection is that the slicing process ensures that for any tuple, there are generally multiple matching buckets. Given a tuple  $t = \langle v_1, v_2, \dots, v_c \rangle$ , where  $c$  is the number of columns and  $v_i$  is the value for the  $i$ th column, a bucket is a matching bucket for  $t$  if and only if for each  $i$  ( $1 \leq i \leq c$ ),  $v_i$  appears at least once in the  $i$ th column of the bucket. Any bucket that contains the original tuple is a matching bucket. At the same time, a matching bucket can be due to containing other tuples each of which contains some but not all  $v_i$ 's.

## 1.2 Contributions & Organization

In this paper, we present a novel technique called *slicing* for privacy-preserving data publishing. Our contributions include the following.

First, we introduce slicing as a new technique for privacy preserving data publishing. Slicing has several advantages when compared with generalization and bucketization. It preserves better data utility than generalization. It preserves more attribute correlations with the SAs than bucketization. It can also handle high-dimensional data and data without a clear separation of QIs and SAs.

Second, we show that slicing can be effectively used for preventing attribute disclosure, based on the privacy requirement of  $\ell$ -diversity. We introduce a notion called  $\ell$ -diverse slicing, which ensures that the adversary cannot learn the sensitive value of *any* individual with a probability greater than  $1/\ell$ .

Third, we develop an efficient algorithm for computing the sliced table that satisfies  $\ell$ -diversity. Our algorithm partitions attributes into columns, applies column generalization, and partitions tuples into buckets. Attributes that are highly correlated are in the same column; this preserves the correlations between such attributes. The associations between uncorrelated attributes are broken; this provides better privacy as the associations between such attributes are less frequent and potentially identifying.

Fourth, we describe the intuition behind membership disclosure and explain how slicing prevents membership disclosure. A bucket of size  $k$  can potentially match  $k^c$  tuples where  $c$  is the number of columns. Because only  $k$  of the  $k^c$  tuples are actually in the original data, the existence of the other  $k^c - k$  tuples hides the membership information of tuples in the original data.

Finally, we conduct extensive workload experiments. Our results confirm that slicing preserves much better data utility than generalization. In workloads involving the sensitive attribute, slicing is also more effective than bucketization. Our experiments also show the limitations of bucketization in membership disclosure protection and slicing remedies these limitations. We also evaluated the performance of slicing in anonymizing the Netflix Prize data set.

The rest of this paper is organized as follows: In Section 2, we formalize the slicing technique and compare it with generalization and bucketization. We define  $\ell$ -diverse slicing for attribute disclosure protection in Section 3 and develop an efficient algorithm to achieve  $\ell$ -diverse slicing in Section 4. In Section 5, we explain how slicing prevents membership disclosure. In Section 6, we present an extensive experimental study on slicing and in Section 7, we evaluate the performance of slicing in anonymizing the Netflix Prize data set. We discuss related work in Section 8 and conclude the paper and discuss future research in Section 9.

## 2 SLICING

In this section, we first give an example to illustrate slicing. We then formalize slicing, compare it with generalization and bucketization, and discuss privacy threats that slicing can address.

Table 1 shows an example microdata table and its anonymized versions using various anonymization techniques. The original table is shown in Table 1a. The three QI attributes are  $\{Age, Sex, Zipcode\}$ , and the sensitive attribute SA is *Disease*. A generalized table that satisfies 4-anonymity is shown in Table 1b, a bucketized table that satisfies 2-diversity is shown in Table 1c, a generalized table where each attribute value is replaced with the multiset of values in the bucket is shown in Table 1d, and two sliced tables are shown in Tables 1e and 1f.

Slicing first partitions attributes into columns. Each column contains a subset of attributes. This vertically partitions the table. For example, the sliced table in Table 1f contains two columns: the first column contains  $\{Age, Sex\}$  and the second column contains  $\{Zipcode, Disease\}$ . The sliced table shown in Table 1e contains four columns, where each column contains exactly one attribute.

Slicing also partitions tuples into buckets. Each bucket contains a subset of tuples. This horizontally partitions the

TABLE 1  
An Original Microdata Table and Its Anonymized Versions Using Various Anonymization Techniques

Age	Sex	Zipcode	Disease
22	M	47906	dyspepsia
22	F	47906	flu
33	F	47905	flu
52	F	47905	bronchitis
54	M	47302	flu
60	M	47302	dyspepsia
60	M	47304	dyspepsia
64	F	47304	gastritis

(a)

Age	Sex	Zipcode	Disease
[20-52]	*	4790*	dyspepsia
[20-52]	*	4790*	flu
[20-52]	*	4790*	flu
[20-52]	*	4790*	bronchitis
[54-64]	*	4730*	flu
[54-64]	*	4730*	dyspepsia
[54-64]	*	4730*	dyspepsia
[54-64]	*	4730*	gastritis

(b)

Age	Sex	Zipcode	Disease
22	M	47906	flu
22	F	47906	dyspepsia
33	F	47905	bronchitis
52	F	47905	flu
54	M	47302	gastritis
60	M	47302	flu
60	M	47304	dyspepsia
64	F	47304	dyspepsia

(c)

Age	Sex	Zipcode	Disease
22:2,33:1,52:1	M:1,F:3	47905:2,47906:2	dysp.
22:2,33:1,52:1	M:1,F:3	47905:2,47906:2	flu
22:2,33:1,52:1	M:1,F:3	47905:2,47906:2	flu
22:2,33:1,52:1	M:1,F:3	47905:2,47906:2	bron.
54:1,60:2,64:1	M:3,F:1	47302:2,47304:2	flu
54:1,60:2,64:1	M:3,F:1	47302:2,47304:2	dysp.
54:1,60:2,64:1	M:3,F:1	47302:2,47304:2	dysp.
54:1,60:2,64:1	M:3,F:1	47302:2,47304:2	gast.

(d)

Age	Sex	Zipcode	Disease
22	F	47906	flu
22	M	47905	flu
33	F	47906	dysp.
52	F	47905	bron.
54	M	47302	dysp.
60	F	47304	gast.
60	M	47302	dysp.
64	M	47304	flu

(e)

(Age,Sex)	(Zipcode,Disease)
(22,M)	(47905,flu)
(22,F)	(47906,dysp.)
(33,F)	(47905,bron.)
(52,F)	(47906,flu)
(54,M)	(47304,gast.)
(60,M)	(47302,flu)
(60,M)	(47302,dysp.)
(64,F)	(47304,dysp.)

(f)

(a) The original table, (b) the generalized table, (c) the bucketized table, (d) multiset-based generalization, (e) one-attribute-per-column slicing, (f) the sliced table.

table. For example, both sliced tables in Tables 1e and 1f contain two buckets, each containing four tuples.

Within each bucket, values in each column are randomly permuted to break the linking between different columns. For example, in the first bucket of the sliced table shown in Table 1f, the values  $\{(22, M), (22, F), (33, F), (52, F)\}$  are randomly permuted and the values  $\{(47906, dyspepsia), (47906, flu), (47905, flu), (47905, bronchitis)\}$  are randomly permuted so that the linking between the two columns within one bucket is hidden.

## 2.1 Formalization of Slicing

Let  $T$  be the microdata table to be published.  $T$  contains  $d$  attributes:  $A = \{A_1, A_2, \dots, A_d\}$  and their attribute domains are  $\{D[A_1], D[A_2], \dots, D[A_d]\}$ . A tuple  $t \in T$  can be represented as  $t = (t[A_1], t[A_2], \dots, t[A_d])$  where  $t[A_i]$  ( $1 \leq i \leq d$ ) is the  $A_i$  value of  $t$ .

**Definition 1 (Attribute Partition and Columns).** An *attribute partition* consists of several subsets of  $A$ , such that each attribute belongs to exactly one subset. Each subset of attributes is called a *column*. Specifically, let there be  $c$  columns  $C_1, C_2, \dots, C_c$ , then  $\cup_{i=1}^c C_i = A$  and for any  $1 \leq i_1 \neq i_2 \leq c$ ,  $C_{i_1} \cap C_{i_2} = \emptyset$ .

For simplicity of discussion, we consider only one sensitive attribute  $S$ . If the data contain multiple sensitive attributes, one can either consider them separately or consider their joint distribution [25]. Exactly one of the  $c$  columns contains  $S$ . Without loss of generality, let the column that contains  $S$  be the last column  $C_c$ . This column is also called the *sensitive column*. All other columns  $\{C_1, C_2, \dots, C_{c-1}\}$  contain only QI attributes.

**Definition 2 (Tuple Partition and Buckets).** A *tuple partition* consists of several subsets of  $T$ , such that each tuple belongs to exactly one subset. Each subset of tuples is called a *bucket*. Specifically, let there be  $b$  buckets

$B_1, B_2, \dots, B_b$ , then  $\cup_{i=1}^b B_i = T$  and for any  $1 \leq i_1 \neq i_2 \leq b$ ,  $B_{i_1} \cap B_{i_2} = \emptyset$ .

**Definition 3 (Slicing).** Given a microdata table  $T$ , a *slicing* of  $T$  is given by an *attribute partition* and a *tuple partition*.

For example, Tables 1e and 1f are two sliced tables. In Table 1e, the attribute partition is  $\{\{Age\}, \{Sex\}, \{Zipcode\}, \{Disease\}\}$  and the tuple partition is  $\{\{t_1, t_2, t_3, t_4\}, \{t_5, t_6, t_7, t_8\}\}$ . In Table 1f, the attribute partition is  $\{\{Age, Sex\}, \{Zipcode, Disease\}\}$  and the tuple partition is  $\{\{t_1, t_2, t_3, t_4\}, \{t_5, t_6, t_7, t_8\}\}$ .

Often times, slicing also involves column generalization.

**Definition 4 (Column Generalization).** Given a microdata table  $T$  and a column  $C_i = \{A_{i1}, A_{i2}, \dots, A_{ij}\}$  where  $A_{i1}, A_{i2}, \dots, A_{ij}$  are attributes, a *column generalization* for  $C_i$  is defined as a set of nonoverlapping  $j$ -dimensional regions that completely cover  $D[A_{i1}] \times D[A_{i2}] \times \dots \times D[A_{ij}]$ . A column generalization maps each value of  $C_i$  to the region in which the value is contained.

Column generalization ensures that one column satisfies the  $k$ -anonymity requirement. It is a multidimensional encoding [19] and can be used as an additional step in slicing. Specifically, a general slicing algorithm consists of the following three phases: attribute partition, column generalization, and tuple partition. Because each column contains much fewer attributes than the whole table, attribute partition enables slicing to handle high-dimensional data.

A key notion of slicing is that of *matching buckets*.

**Definition 5 (Matching Buckets).** Let  $\{C_1, C_2, \dots, C_c\}$  be the  $c$  columns of a sliced table. Let  $t$  be a tuple, and  $t[C_i]$  be the  $C_i$  value of  $t$ . Let  $B$  be a bucket in the sliced table, and  $B[C_i]$  be the multiset of  $C_i$  values in  $B$ . We say that  $B$  is a *matching bucket* of  $t$  iff for all  $1 \leq i \leq c$ ,  $t[C_i] \in B[C_i]$ .

For example, consider the sliced table shown in Table 1f, and consider  $t_1 = (22, M, 47906, dyspepsia)$ . Then, the set of matching buckets for  $t_1$  is  $\{B_1\}$ .

## 2.2 Comparison with Generalization

There are several types of recodings for generalization. The recoding that preserves the most information is *local recoding* [36]. In local recoding, one first groups tuples into buckets and then for each bucket, one replaces all values of one attribute with a generalized value. Such a recoding is local because the same attribute value may be generalized differently when they appear in different buckets.

We now show that slicing preserves more information than such a local recoding approach, assuming that the same tuple partition is used. We achieve this by showing that slicing is better than the following enhancement of the local recoding approach. Rather than using a generalized value to replace more specific attribute values, one uses the multiset of exact values in each bucket. For example, Table 1b is a generalized table, and Table 1d is the result of using multisets of exact values rather than generalized values. For the *Age* attribute of the first bucket, we use the multiset of exact values  $\{22, 22, 33, 52\}$  rather than the generalized interval  $[22-52]$ . The multiset of exact values provides more information about the distribution of values in each attribute than the generalized interval. Therefore, using multisets of exact values preserves more information than generalization.

However, we observe that this multiset-based generalization is equivalent to a trivial slicing scheme where each column contains exactly one attribute, because both approaches preserve the exact values in each attribute but break the association between them within one bucket. For example, Table 1e is equivalent to Table 1d. Now comparing Table 1e with the sliced table shown in Table 1f, we observe that while one-attribute-per-column slicing preserves attribute distributional information, it does not preserve attribute correlation, because each attribute is in its own column. In slicing, one groups correlated attributes together in one column and preserves their correlation. For example, in the sliced table shown in Table 1f, correlations between *Age* and *Sex* and correlations between *Zipcode* and *Disease* are preserved. In fact, the sliced table encodes the same amount of information as the original data with regard to correlations between attributes in the same column.

Another important advantage of slicing is its ability to handle high-dimensional data. By partitioning attributes into columns, slicing reduces the dimensionality of the data. Each column of the table can be viewed as a subtable with a lower dimensionality. Slicing is also different from the approach of publishing multiple independent subtables [16] in that these subtables are linked by the buckets in slicing. The idea of slicing is to achieve a better trade-off between privacy and utility by preserving correlations between highly correlated attributes and breaking correlations between uncorrelated attributes.

## 2.3 Comparison with Bucketization

To compare slicing with bucketization, we first note that bucketization can be viewed as a special case of slicing, where there are exactly two columns: one column contains

only the SA, and the other contains all the QIs. The advantages of slicing over bucketization can be understood as follows: First, by partitioning attributes into more than two columns, slicing can be used to prevent membership disclosure. Our empirical evaluation on a real data set shows that bucketization does not prevent membership disclosure in Section 6.

Second, unlike bucketization, which requires a clear separation of QI attributes and the sensitive attribute, slicing can be used without such a separation. For data set such as the census data, one often cannot clearly separate QIs from SAs because there is no single external public database that one can use to determine which attributes the adversary already knows. Slicing can be useful for such data.

Finally, by allowing a column to contain both some QI attributes and the sensitive attribute, attribute correlations between the sensitive attribute and the QI attributes are preserved. For example, in Table 1f, *Zipcode* and *Disease* form one column, enabling inferences about their correlations. Attribute correlations are important utility in data publishing. For workloads that consider attributes in isolation, one can simply publish two tables, one containing all QI attributes and one containing the sensitive attribute.

## 2.4 Privacy Threats

When publishing microdata, there are three types of privacy disclosure threats. The first type is *membership disclosure*. When the data set to be published is selected from a large population and the selection criteria are sensitive (e.g., only diabetes patients are selected), one needs to prevent adversaries from learning whether one's record is included in the published data set.

The second type is *identity disclosure*, which occurs when an individual is linked to a particular record in the released table. In some situations, one wants to protect against identity disclosure when the adversary is uncertain of membership. In this case, protection against membership disclosure helps protect against identity disclosure. In other situations, some adversary may already know that an individual's record is in the published data set, in which case, membership disclosure protection either does not apply or is insufficient.

The third type is *attribute disclosure*, which occurs when new information about some individuals is revealed, i.e., the released data make it possible to infer the attributes of an individual more accurately than it would be possible before the release. Similar to the case of identity disclosure, we need to consider adversaries who already know the membership information. Identity disclosure leads to attribute disclosure. Once there is identity disclosure, an individual is reidentified and the corresponding sensitive value is revealed. Attribute disclosure can occur with or without identity disclosure, e.g., when the sensitive values of all matching tuples are the same.

For slicing, we consider protection against membership disclosure and attribute disclosure. It is a little unclear how identity disclosure should be defined for sliced data (or for data anonymized by bucketization), since each tuple resides within a bucket and within the bucket the association across different columns are hidden. In any case, because identity disclosure leads to attribute disclosure, protection against

attribute disclosure is also sufficient protection against identity disclosure.

We would like to point out a nice property of slicing that is important for privacy protection. In slicing, a tuple can potentially match multiple buckets, i.e., each tuple can have more than one matching buckets. This is different from previous work on generalization (global recoding specifically) and bucketization, where each tuple can belong to a unique equivalence-class (or bucket). In fact, it has been recognized [3] that restricting a tuple in a unique bucket helps the adversary but does not improve data utility. We will see that allowing a tuple to match multiple buckets is important for both attribute disclosure protection and membership disclosure protection, when we describe them in Sections 3 and 5, respectively.

### 3 ATTRIBUTE DISCLOSURE PROTECTION

In this section, we show how slicing can be used to prevent attribute disclosure, based on the privacy requirement of  $\ell$ -diversity and introduce the notion of  $\ell$ -diverse slicing.

#### 3.1 Example

We first give an example illustrating how slicing satisfies  $\ell$ -diversity [25] where the sensitive attribute is “Disease.” The sliced table shown in Table 1f satisfies 2-diversity. Consider tuple  $t_1$  with QI values (22,  $M$ , 47906). In order to determine  $t_1$ ’s sensitive value, one has to examine  $t_1$ ’s matching buckets. By examining the first column (*Age*, *Sex*) in Table 1f, we know that  $t_1$  must be in the first bucket  $B_1$  because there are no matches of (22,  $M$ ) in bucket  $B_2$ . Therefore, one can conclude that  $t_1$  cannot be in bucket  $B_2$  and  $t_1$  must be in bucket  $B_1$ .

Then, by examining the *Zipcode* attribute of the second column (*Zipcode*, *Disease*) in bucket  $B_1$ , we know that the column value for  $t_1$  must be either (47906, *dyspepsia*) or (47906, *flu*) because they are the only values that match  $t_1$ ’s zipcode 47906. Note that the other two column values have zipcode 47905. Without additional knowledge, both *dyspepsia* and *flu* are equally possible to be the sensitive value of  $t_1$ . Therefore, the probability of learning the correct sensitive value of  $t_1$  is bounded by 0.5. Similarly, we can verify that 2-diversity is satisfied for all other tuples in Table 1f.

#### 3.2 $\ell$ -Diverse Slicing

In the above example, tuple  $t_1$  has only one matching bucket. In general, a tuple  $t$  can have multiple matching buckets. We now extend the above analysis to the general case and introduce the notion of  $\ell$ -diverse slicing.

Consider an adversary who knows all the QI values of  $t$  and attempts to infer  $t$ ’s sensitive value from the sliced table. She or he first needs to determine which buckets  $t$  may reside in, i.e., the set of matching buckets of  $t$ . Tuple  $t$  can be in any one of its matching buckets. Let  $p(t, B)$  be the probability that  $t$  is in bucket  $B$  (the procedure for computing  $p(t, B)$  will be described later in this section). For example, in the above example,  $p(t_1, B_1) = 1$  and  $p(t_1, B_2) = 0$ .

In the second step, the adversary computes  $p(t, s)$ , the probability that  $t$  takes a sensitive value  $s$ .  $p(t, s)$  is

calculated using *the law of total probability*. Specifically, let  $p(s|t, B)$  be the probability that  $t$  takes sensitive value  $s$  given that  $t$  is in bucket  $B$ , then according to the law of total probability, the probability  $p(t, s)$  is

$$p(t, s) = \sum_B p(t, B)p(s|t, B). \quad (1)$$

In the rest of this section, we show how to compute the two probabilities:  $p(t, B)$  and  $p(s|t, B)$ .

**Computing  $p(t, B)$ .** Given a tuple  $t$  and a sliced bucket  $B$ , the probability that  $t$  is in  $B$  depends on the fraction of  $t$ ’s column values that match the column values in  $B$ . If some column value of  $t$  does not appear in the corresponding column of  $B$ , it is certain that  $t$  is not in  $B$ . In general, bucket  $B$  can potentially match  $|B|^c$  tuples, where  $|B|$  is the number of tuples in  $B$ . Without additional knowledge, one has to assume that the column values are independent; therefore, each of the  $|B|^c$  tuples is equally likely to be an original tuple. The probability that  $t$  is in  $B$  depends on the fraction of the  $|B|^c$  tuples that match  $t$ .

We formalize the above analysis. We consider the match between  $t$ ’s column values  $\{t[C_1], t[C_2], \dots, t[C_c]\}$  and  $B$ ’s column values  $\{B[C_1], B[C_2], \dots, B[C_c]\}$ . Let  $f_i(t, B)$  ( $1 \leq i \leq c-1$ ) be the fraction of occurrences of  $t[C_i]$  in  $B[C_i]$  and let  $f_c(t, B)$  be the fraction of occurrences of  $t[C_c - \{S\}]$  in  $B[C_c - \{S\}]$ . Note that,  $C_c - \{S\}$  is the set of QI attributes in the sensitive column. For example, in Table 1f,  $f_1(t_1, B_1) = 1/4 = 0.25$  and  $f_2(t_1, B_1) = 2/4 = 0.5$ . Similarly,  $f_1(t_1, B_2) = 0$  and  $f_2(t_1, B_2) = 0$ . Intuitively,  $f_i(t, B)$  measures the *matching degree* on column  $C_i$ , between tuple  $t$  and bucket  $B$ .

Because each possible candidate tuple is equally likely to be an original tuple, the *matching degree* between  $t$  and  $B$  is the product of the matching degree on each column, i.e.,  $f(t, B) = \prod_{1 \leq i \leq c} f_i(t, B)$ . Note that  $\sum_t f(t, B) = 1$  and when  $B$  is not a matching bucket of  $t$ ,  $f(t, B) = 0$ .

Tuple  $t$  may have multiple matching buckets,  $t$ ’s total matching degree in the whole data is  $f(t) = \sum_B f(t, B)$ . The probability that  $t$  is in bucket  $B$  is

$$p(t, B) = \frac{f(t, B)}{f(t)}.$$

**Computing  $p(s|t, B)$ .** Suppose that  $t$  is in bucket  $B$ , to determine  $t$ ’s sensitive value, one needs to examine the sensitive column of bucket  $B$ . Since the sensitive column contains the QI attributes, not all sensitive values can be  $t$ ’s sensitive value. Only those sensitive values whose QI values match  $t$ ’s QI values are  $t$ ’s *candidate sensitive values*. Without additional knowledge, all candidate sensitive values (including duplicates) in a bucket are equally possible. Let  $D(t, B)$  be the distribution of  $t$ ’s candidate sensitive values in bucket  $B$ .

**Definition 6 ( $D(t, B)$ ).** Any sensitive value that is associated with  $t[C_c - \{S\}]$  in  $B$  is a *candidate sensitive value* for  $t$  (there are  $f_c(t, B)$  candidate sensitive values for  $t$  in  $B$ , including duplicates). Let  $D(t, B)$  be the distribution of the candidate sensitive values in  $B$  and  $D(t, B)[s]$  be the probability of the sensitive value  $s$  in the distribution.

For example, in Table 1f,  $D(t_1, B_1) = (\text{dyspepsia} : 0.5, \text{flu} : 0.5)$  and therefore  $D(t_1, B_1)[\text{dyspepsia}] = 0.5$ . The probability  $p(s|t, B)$  is exactly  $D(t, B)[s]$ , i.e.,  $p(s|t, B) = D(t, B)[s]$ .

**$\ell$ -Diverse Slicing.** Once we have computed  $p(t, B)$  and  $p(s|t, B)$ , we are able to compute the probability  $p(t, s)$  based on the (1). We can show when  $t$  is in the data, the probabilities that  $t$  takes a sensitive value sum up to 1.

**Fact 1.** For any tuple  $t \in D$ ,  $\sum_s p(t, s) = 1$ .

**Proof.**

$$\begin{aligned} \sum_s p(t, s) &= \sum_s \sum_B p(t, B) p(s|t, B) \\ &= \sum_B p(t, B) \sum_s p(s|t, B) \\ &= \sum_B p(t, B) \\ &= 1. \end{aligned} \quad (2)$$

□

$\ell$ -Diverse slicing is defined based on the probability  $p(t, s)$ .

**Definition 7 ( $\ell$ -diverse slicing).** A tuple  $t$  satisfies  $\ell$ -diversity iff for any sensitive value  $s$

$$p(t, s) \leq 1/\ell.$$

A sliced table satisfies  $\ell$ -diversity iff every tuple in it satisfies  $\ell$ -diversity.

Our analysis above directly show that from an  $\ell$ -diverse sliced table, an adversary cannot correctly learn the sensitive value of any individual with a probability greater than  $1/\ell$ . Note that once we have computed the probability that a tuple takes a sensitive value, we can also use slicing for other privacy measures such as  $t$ -closeness [21].

## 4 SLICING ALGORITHMS

We now present an efficient slicing algorithm to achieve  $\ell$ -diverse slicing. Given a microdata table  $T$  and two parameters  $c$  and  $\ell$ , the algorithm computes the sliced table that consists of  $c$  columns and satisfies the privacy requirement of  $\ell$ -diversity.

Our algorithm consists of three phases: *attribute partitioning*, *column generalization*, and *tuple partitioning*. We now describe the three phases.

### 4.1 Attribute Partitioning

Our algorithm partitions attributes so that highly correlated attributes are in the same column. This is good for both utility and privacy. In terms of data utility, grouping highly correlated attributes preserves the correlations among those attributes. In terms of privacy, the association of uncorrelated attributes presents higher identification risks than the association of highly correlated attributes because the association of uncorrelated attribute values is much less frequent and thus more identifiable. Therefore, it is better to break the associations between uncorrelated attributes, in order to protect privacy.

In this phase, we first compute the correlations between pairs of attributes and then cluster attributes based on their correlations.

#### 4.1.1 Measures of Correlation

Two widely used measures of association are Pearson correlation coefficient [5] and mean-square contingency coefficient [5]. Pearson correlation coefficient is used for measuring correlations between two continuous attributes while mean-square contingency coefficient is a chi-square measure of correlation between two categorical attributes. We choose to use the *mean-square contingency coefficient* because most of our attributes are categorical. Given two attributes  $A_1$  and  $A_2$  with domains  $\{v_{11}, v_{12}, \dots, v_{1d_1}\}$  and  $\{v_{21}, v_{22}, \dots, v_{2d_2}\}$ , respectively. Their domain sizes are thus  $d_1$  and  $d_2$ , respectively. The mean-square contingency coefficient between  $A_1$  and  $A_2$  is defined as

$$\phi^2(A_1, A_2) = \frac{1}{\min\{d_1, d_2\} - 1} \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \frac{(f_{ij} - f_{i.} f_{.j})^2}{f_{i.} f_{.j}}.$$

Here,  $f_{i.}$  and  $f_{.j}$  are the fraction of occurrences of  $v_{1i}$  and  $v_{2j}$  in the data, respectively.  $f_{ij}$  is the fraction of cooccurrences of  $v_{1i}$  and  $v_{2j}$  in the data. Therefore,  $f_{i.}$  and  $f_{.j}$  are the marginal totals of  $f_{ij}$ :  $f_{i.} = \sum_{j=1}^{d_2} f_{ij}$  and  $f_{.j} = \sum_{i=1}^{d_1} f_{ij}$ . It can be shown that  $0 \leq \phi^2(A_1, A_2) \leq 1$ .

For continuous attributes, we first apply *discretization* to partition the domain of a continuous attribute into intervals and then treat the collection of interval values as a discrete domain. Discretization has been frequently used for decision tree classification, summarization, and frequent item set mining. We use equal-width discretization, which partitions an attribute domain into (some  $k$ ) equal-sized intervals. Other methods for handling continuous attributes are the subjects of future work.

#### 4.1.2 Attribute Clustering

Having computed the correlations for each pair of attributes, we use clustering to partition attributes into columns. In our algorithm, each attribute is a point in the clustering space. The distance between two attributes in the clustering space is defined as  $d(A_1, A_2) = 1 - \phi^2(A_1, A_2)$ , which is in between of 0 and 1. Two attributes that are strongly correlated will have a smaller distance between the corresponding data points in our clustering space.

We choose the  $k$ -medoid method for the following reasons. First, many existing clustering algorithms (e.g.,  $k$ -means) requires the calculation of the “centroids.” But there is no notion of “centroids” in our setting where each attribute forms a data point in the clustering space. Second,  $k$ -medoid method is very robust to the existence of outliers (i.e., data points that are very far away from the rest of data points). Third, the order in which the data points are examined does not affect the clusters computed from the  $k$ -medoid method.  $k$ -Medoid is NP-hard in general and we use the well-known  $k$ -medoid algorithm PAM (Partition Around Medoids) [15]. PAM starts by an arbitrary selection of  $k$  data points as the initial medoids. In each subsequent step, PAM chooses one medoid point and one nonmedoid point and swaps them as long as the cost of clustering decreases. Here, the clustering cost is measured as the sum

**Algorithm tuple-partition( $T, \ell$ )**

1.  $Q = \{T\}$ ;  $SB = \emptyset$ .
2. while  $Q$  is not empty
3.   remove the first bucket  $B$  from  $Q$ ;  $Q = Q - \{B\}$ .
4.   split  $B$  into two buckets  $B_1$  and  $B_2$ , as in Mondrian.
5.   if **diversity-check**( $T, Q \cup \{B_1, B_2\} \cup SB, \ell$ )
6.      $Q = Q \cup \{B_1, B_2\}$ .
7.   else  $SB = SB \cup \{B\}$ .
8. return  $SB$ .

Fig. 1. The tuple-partition algorithm.

of the cost of each cluster, which is in turn measured as the sum of the distance from each data point in the cluster to the medoid point of the cluster. The time complexity of PAM is  $O(k(m - k)^2)$ . Thus, it is known that PAM suffers from high-computational complexity for large data sets. However, the data points in our clustering space are attributes, rather than tuples in the microdata. Therefore, PAM will not have computational problems for clustering attributes.

#### 4.1.3 Special Attribute Partitioning

In the above procedure, all attributes (including both QIs and SAs) are clustered into columns. The  $k$ -medoid method ensures that the attributes are clustered into  $k$  columns but does not have any guarantee on the size of the sensitive column  $C_c$ . In some cases, we may predetermine the number of attributes in the sensitive column to be  $\alpha$ . The parameter  $\alpha$  determines the size of the sensitive column  $C_c$ , i.e.,  $|C_c| = \alpha$ . If  $\alpha = 1$ , then  $|C_c| = 1$ , which means that  $C_c = \{S\}$ . And when  $c = 2$ , slicing in this case becomes equivalent to bucketization. If  $\alpha > 1$ , then  $|C_c| > 1$ , the sensitive column also contains some QI attributes.

We adapt the above algorithm to partition attributes into  $c$  columns such that the sensitive column  $C_c$  contains  $\alpha$  attributes. We first calculate correlations between the sensitive attribute  $S$  and each QI attribute. Then, we rank the QI attributes by the decreasing order of their correlations with  $S$  and select the top  $\alpha - 1$  QI attributes. Now, the sensitive column  $C_c$  consists of  $S$  and the selected QI attributes. All other QI attributes form the other  $c - 1$  columns using the attribute clustering algorithm.

#### 4.2 Column Generalization

In the second phase, tuples are generalized to satisfy some minimal frequency requirement. We want to point out that column generalization is not an indispensable phase in our algorithm. As shown by Xiao and Tao [34], bucketization provides the same level of privacy protection as generalization, with respect to attribute disclosure.

Although column generalization is not a required phase, it can be useful in several aspects. First, column generalization may be required for identity/membership disclosure protection. If a column value is unique in a column (i.e., the column value appears only once in the column), a tuple with this unique column value can only have one matching bucket. This is not good for privacy protection, as in the case of generalization/bucketization where each tuple can belong to only one equivalence-class/bucket. The main problem is that this unique column value can be

**Algorithm diversity-check( $T, T^*, \ell$ )**

1. for each tuple  $t \in T$ ,  $L[t] = \emptyset$ .
2. for each bucket  $B$  in  $T^*$
3.   record  $f(v)$  for each column value  $v$  in bucket  $B$ .
4.   for each tuple  $t \in T$
5.     calculate  $p(t, B)$  and find  $D(t, B)$ .
6.      $L[t] = L[t] \cup \{p(t, B), D(t, B)\}$ .
7. for each tuple  $t \in T$
8.   calculate  $p(t, s)$  for each  $s$  based on  $L[t]$ .
9.   if  $p(t, s) \geq 1/\ell$ , return false.
10. return true.

Fig. 2. The diversity-check algorithm.

identifying. In this case, it would be useful to apply column generalization to ensure that each column value appears with at least some frequency.

Second, when column generalization is applied, to achieve the same level of privacy against attribute disclosure, bucket sizes can be smaller (see Section 4.3). While column generalization may result in information loss, smaller bucket-sizes allow better data utility. Therefore, there is a trade-off between column generalization and tuple partitioning. In this paper, we mainly focus on the tuple partitioning algorithm. The trade-off between column generalization and tuple partitioning is the subject of future work. Existing anonymization algorithms can be used for column generalization, e.g., Mondrian [19]. The algorithms can be applied on the subtable containing only attributes in one column to ensure the anonymity requirement.

#### 4.3 Tuple Partitioning

In the tuple partitioning phase, tuples are partitioned into buckets. We modify the Mondrian [19] algorithm for tuple partition. Unlike Mondrian  $k$ -anonymity, no generalization is applied to the tuples; we use Mondrian for the purpose of partitioning tuples into buckets.

Fig. 1 gives the description of the tuple-partition algorithm. The algorithm maintains two data structures: 1) a queue of buckets  $Q$  and 2) a set of sliced buckets  $SB$ . Initially,  $Q$  contains only one bucket which includes all tuples and  $SB$  is empty (line 1). In each iteration (lines 2 to 7), the algorithm removes a bucket from  $Q$  and splits the bucket into two buckets (the split criteria is described in Mondrian [19]). If the sliced table after the split satisfies  $\ell$ -diversity (line 5), then the algorithm puts the two buckets at the end of the queue  $Q$  (for more splits, line 6). Otherwise, we cannot split the bucket anymore and the algorithm puts the bucket into  $SB$  (line 7). When  $Q$  becomes empty, we have computed the sliced table. The set of sliced buckets is  $SB$  (line 8).

The main part of the tuple-partition algorithm is to check whether a sliced table satisfies  $\ell$ -diversity (line 5). Fig. 2 gives a description of the *diversity-check* algorithm. For each tuple  $t$ , the algorithm maintains a list of statistics  $L[t]$  about  $t$ 's matching buckets. Each element in the list  $L[t]$  contains statistics about one matching bucket  $B$ : the matching probability  $p(t, B)$  and the distribution of candidate sensitive values  $D(t, B)$ .



The algorithm first takes one scan of each bucket  $B$  (lines 2 to 3) to record the frequency  $f(v)$  of each column value  $v$  in bucket  $B$ . Then, the algorithm takes one scan of each tuple  $t$  in the table  $T$  (lines 4 to 6) to find out all tuples that match  $B$  and record their matching probability  $p(t, B)$  and the distribution of candidate sensitive values  $D(t, B)$ , which are added to the list  $L[t]$  (line 6). At the end of line 6, we have obtained, for each tuple  $t$ , the list of statistics  $L[t]$  about its matching buckets. A final scan of the tuples in  $T$  will compute the  $p(t, s)$  values based on the *law of total probability* described in Section 3.2. Specifically

$$p(t, s) = \sum_{e \in L[t]} e.p(t, B) * e.D(t, B)[s].$$

The sliced table is  $\ell$ -diverse iff for all sensitive value  $s$ ,  $p(t, s) \leq 1/\ell$  (lines 7 to 10).

We now analyze the time complexity of the tuple-partition algorithm. The time complexity of Mondrian [19] or kd-tree [10] is  $O(n \log n)$  because at each level of the kd-tree, the whole data set need to be scanned which takes  $O(n)$  time and the height of the tree is  $O(\log n)$ . In our modification, each level takes  $O(n^2)$  time because of the diversity-check algorithm (note that the number of buckets is at most  $n$ ). The total time complexity is therefore  $O(n^2 \log n)$ .

## 5 MEMBERSHIP DISCLOSURE PROTECTION

In this section, we analyze how slicing can provide membership disclosure protection.

**Bucketization.** Let us first examine how an adversary can infer membership information from bucketization. Because bucketization releases each tuple's combination of QI values in their original form and most individuals can be uniquely identified using the QI values, the adversary can determine the membership of an individual in the original data by examining whether the individual's combination of QI values occurs in the released data.

**Slicing.** Slicing offers protection against membership disclosure because QI attributes are partitioned into different columns and correlations among different columns within each bucket are broken. Consider the sliced table in Table 1f. The table has two columns. The first bucket is resulted from four tuples; we call them the *original tuples*. The bucket matches altogether  $4^2 = 16$  tuples, 4 original tuples and 12 that do not appear in the original table. We call these 12 tuples *fake tuples*. Given any tuple, if it has no matching bucket in the sliced table, then we know for sure that the tuple is not in the original table. However, even if a tuple has one or more matching bucket, one cannot tell whether the tuple is in the original table, because it could be a fake tuple.

We propose two quantitative measures for the degree of membership protection offered by slicing. The first is the fake-original ratio (FOR), which is defined as the number of fake tuples divided by the number of original tuples. Intuitively, the larger the FOR, the more membership protection is provided. A sliced bucket of size  $k$  can potentially match  $k^c$  tuples, including  $k$  original tuples and  $k^c - k$  fake tuples; hence, the FOR is  $k^{c-1} - 1$ . When one

has chosen a minimal threshold for the FOR, one can choose  $k$  and  $c$  appropriately to satisfy the threshold.

The second measure is to consider the number of matching buckets for original tuples and that for fake tuples. If they are similar enough, membership information is protected because the adversary cannot distinguish original tuples from fake tuples. Since the main focus of this paper is attribute disclosure, we do not intend to propose a comprehensive analysis for membership disclosure protection. In our experiments (Section 6), we empirically compare bucketization and slicing in terms of the number of matching buckets for tuples that are in or not in the original data. Our experimental results show that slicing introduces a large number of tuples in D and can be used to protect membership information.

**Generalization.** By generalizing attribute values into "less-specific but semantically consistent values," generalization offers some protection against membership disclosure. It was shown in [27] that generalization alone (e.g., used with  $k$ -anonymity) may leak membership information if the target individual is the only possible match for a generalized record. The intuition is similar to our rationale of fake tuple. If a generalized tuple does not introduce fake tuples (i.e., none of the other combinations of values are reasonable), there will be only one original tuple that matches with the generalized tuple and the membership information can still be inferred. Nergiz et al. [27] defined a large background table as the set of all "possible" tuples in order to estimate the probability whether a tuple is in the data or not ( $\delta$ -presence). The major problem with [27] is that it can be difficult to define the background table and in some cases the data publisher may not have such a background table. Also, the protection against membership disclosure depends on the choice of the background table. Therefore, with careful anonymization, generalization can offer some level of membership disclosure protection.

## 6 EXPERIMENTS

We conduct three experiments. In the first experiment, we evaluate the effectiveness of slicing in preserving data utility and protecting against attribute disclosure, as compared to generalization and bucketization. To allow direct comparison, we use the Mondrian algorithm [19] and  $\ell$ -diversity for all three anonymization techniques: generalization, bucketization, and slicing. This experiment demonstrates that: 1) slicing preserves better data utility than generalization; 2) slicing is more effective than bucketization in workloads involving the sensitive attribute; and 3) the sliced table can be computed efficiently. Results for this experiment are presented in Section 6.2.

In the second experiment, we show the effectiveness of slicing in membership disclosure protection. For this purpose, we count the number of fake tuples in the sliced data. We also compare the number of matching buckets for original tuples and that for fake tuples. Our experimental results show that bucketization does not prevent membership disclosure as almost every tuple is uniquely identifiable in the bucketized data. Slicing provides better protection against membership disclosure: 1) the number of fake tuples in the sliced data is very large, as compared to



TABLE 2  
Description of the **Adult** Data Set

	Attribute	Type	# of values
1	Age	Continuous	74
2	Workclass	Categorical	8
3	Final-Weight	Continuous	NA
4	Education	Categorical	16
5	Education-Num	Continuous	16
6	Marital-Status	Categorical	7
7	Occupation	Categorical	14
8	Relationship	Categorical	6
9	Race	Categorical	5
10	Sex	Categorical	2
11	Capital-Gain	Continuous	NA
12	Capital-Loss	Continuous	NA
13	Hours-Per-Week	Continuous	NA
14	Country	Categorical	41
15	Salary	Categorical	2

the number of original tuples and 2) the number of matching buckets for fake tuples and that for original tuples are close enough, which makes it difficult for the adversary to distinguish fake tuples from original tuples. Results for this experiment are presented in Section 6.3.

**Experimental Data.** We used the Adult data set from the UC Irvine machine learning repository,<sup>1</sup> which is comprised of data collected from the US census. The data set is described in Table 2. Tuples with missing values are eliminated and there are 45,222 valid tuples in total. The adult data set contains 15 attributes in total.

In our experiments, we obtain two data sets from the Adult data set. The first data set is the “OCC-7” data set, which includes seven attributes:  $QI = \{Age, Workclass, Education, Marital-Status, Race, Sex\}$  and  $S = Occupation$ . The second data set is the “OCC-15” data set, which includes all 15 attributes and the sensitive attribute is  $S = Occupation$ . Note that we do not use *Salary* as the sensitive attribute because *Salary* has only two values  $\{\geq 50K, < 50K\}$ , which means that even 2-diversity is not achievable when the sensitive attribute is *Salary*. Also note that in membership disclosure protection, we do not differentiate between QIs and SA.

In the “OCC-7” data set, the attribute that has the closest correlation with the sensitive attribute *Occupation* is *Gender*, with the next closest attribute being *Education*. In the “OCC-15” data set, the closest attribute is also *Gender* but the next closest attribute is *Salary*.

## 6.1 Preprocessing

Some preprocessing steps must be applied on the anonymized data before it can be used for workload tasks. In particular, the anonymized table computed by bucketization or slicing contains multiple columns, the linking between which is broken. We need to process such data before workload experiments can run on the data.

**Handling bucketized/sliced data.** In both bucketization and slicing, attributes are partitioned into two or more columns. For a bucket that contains  $k$  tuples and  $c$  columns, we generate  $k$  tuples as follows: We first randomly permute the values in each column. Then, we generate

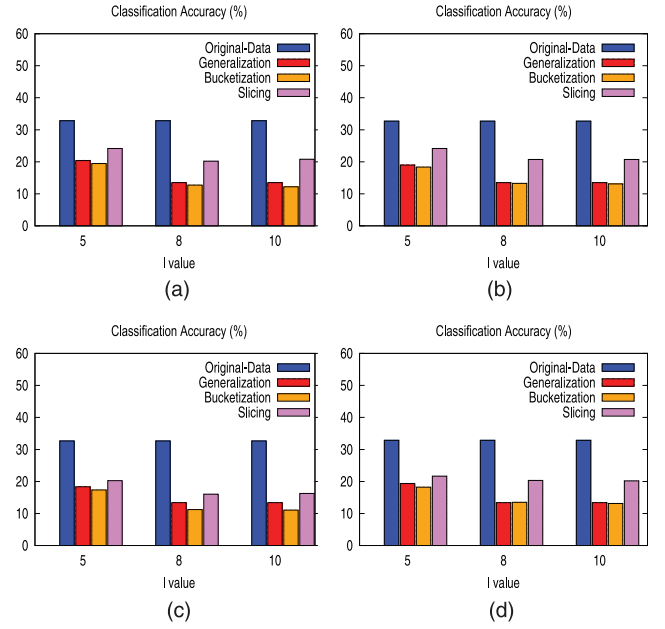


Fig. 3. Learning the sensitive attribute (Target: Occupation). (a) J48 (OCC-7), (b) Naive Bayes (OCC-7), (c) J48 (OCC-15), and (d) Naive Bayes (OCC-15).

the  $i$ th ( $1 \leq i \leq k$ ) tuple by linking the  $i$ th value in each column. We apply this procedure to all buckets and generate all of the tuples from the bucketized/sliced table. This procedure generates the linking between the two columns in a random fashion. In all of our classification experiments, we apply this procedure 5 times and the average results are reported.

## 6.2 Attribute Disclosure Protection

We compare slicing with generalization and bucketization on data utility of the anonymized data for classifier learning. For all three techniques, we employ the Mondrian algorithm [19] to compute the  $\ell$ -diverse tables. The  $\ell$  value can take values  $\{5, 8, 10\}$  (note that the *Occupation* attribute has 14 distinct values). In this experiment, we choose  $\alpha = 2$ . Therefore, the sensitive column is always  $\{Gender, Occupation\}$ .

**Classifier learning.** We evaluate the quality of the anonymized data for classifier learning, which has been used in [11], [20], [3]. We use the Weka software package to evaluate the classification accuracy for Decision Tree C4.5 (J48) and Naive Bayes. Default settings are used in both tasks. For all classification experiments, we use 10-fold cross validation.

In our experiments, we choose one attribute as the target attribute (the attribute on which the classifier is built) and all other attributes serve as the predictor attributes. We consider the performances of the anonymization algorithms in both learning the sensitive attribute *Occupation* and learning a QI attribute *Education*.

**Learning the sensitive attribute.** In this experiment, we build a classifier on the sensitive attribute, which is “*Occupation*.” We fix  $c = 2$  here and evaluate the effects of  $c$  later in this section. In other words, the target attribute is *Occupation* and all other attributes are predictor attributes. Fig. 3 compares the quality of the anonymized

1. <http://archive.ics.uci.edu/ml/>.

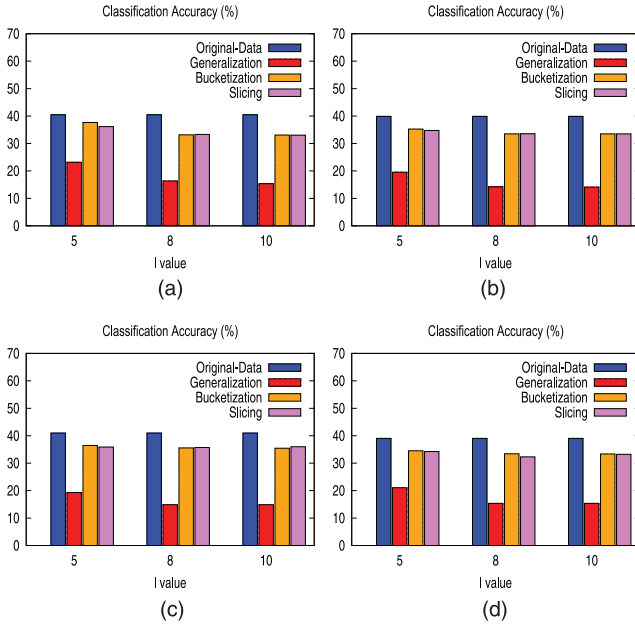


Fig. 4. Learning a QI attribute (Target: Education). (a) J48 (OCC-7), (b) Naive Bayes (OCC-7), (c) J48 (OCC-15), and (d) Naive Bayes (OCC-15).

data (generated by the three techniques) with the quality of the original data, when the target attribute is *Occupation*. The experiments are performed on the two data sets OCC-7 (with seven attributes) and OCC-15 (with 15 attributes). Fig. 3a (Fig. 3b) shows the classification accuracy of J48 (Naive Bayes) on the original data and the three anonymization techniques as a function of the  $\ell$  value for the OCC-7 data set. Fig. 3c (Fig. 3d) shows the results for the OCC-15 data set.

In all experiments, slicing outperforms both generalization and bucketization, that confirms that slicing preserves attribute correlations between the sensitive attribute and some QIs (recall that the sensitive column is  $\{Gender, Occupation\}$ ). Another observation is that bucketization performs even slightly worse than generalization. That is mostly due to our preprocessing step that randomly associates the sensitive values to the QI values in each bucket. This may introduce false associations while in generalization, the associations are always correct although the exact associations are hidden. A final observation is that when  $\ell$  increases, the performances of generalization and bucketization deteriorate much faster than slicing. This also confirms that slicing preserves better data utility in workloads involving the sensitive attribute.

**Learning a QI attribute.** In this experiment, we build a classifier on the QI attribute “Education.” We fix  $c = 2$  here and evaluate the effects of  $c$  later in this section. In other words, the target attribute is *Education* and all other attributes including the sensitive attribute *Occupation* are predictor attributes. Fig. 4 shows the experiment results. Fig. 4a (Fig. 4b) shows the classification accuracy of J48 (Naive Bayes) on the original data and the three anonymization techniques as a function of the  $\ell$  value for the OCC-7 data set. Fig. 4c (Fig. 4d) shows the results for the data set OCC-15.

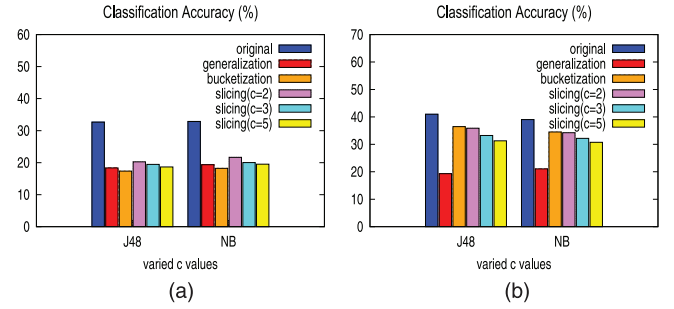


Fig. 5. Varied  $c$  values. (a) Sensitive (OCC-15) and (b) QI (OCC-15).

In all experiments, both bucketization and slicing perform much better than generalization. This is because in both bucketization and slicing, the QI attribute *Education* is in the same column with many other QI attributes: in bucketization, all QI attributes are in the same column; in slicing, all QI attributes except *Gender* are in the same column. This fact allows both approaches to perform well in workloads involving the QI attributes. Note that the classification accuracies of bucketization and slicing are lower than that of the original data. This is because the sensitive attribute *Occupation* is closely correlated with the target attribute *Education* (as mentioned earlier in Section 6, *Education* is the second closest attribute with *Occupation* in OCC-7). By breaking the link between *Education* and *Occupation*, classification accuracy on *Education* reduces for both bucketization and slicing.

**The effects of  $c$ .** In this experiment, we evaluate the effect of  $c$  on classification accuracy. We fix  $\ell = 5$  and vary the number of columns  $c$  in  $\{2, 3, 5\}$ . Fig. 5a shows the results on learning the sensitive attribute and Fig. 5b shows the results on learning a QI attribute. It can be seen that classification accuracy decreases only slightly when we increase  $c$ , because the most correlated attributes are still in the same column. In all cases, slicing shows better accuracy than generalization. When the target attribute is the sensitive attribute, slicing even performs better than bucketization.

**Computational efficiency.** We compare slicing with generalization and bucketization in terms of computational efficiency. We fix  $\ell = 5$  and vary the cardinality of the data (i.e., the number of records) and the dimensionality of the data (i.e., the number of attributes). Fig. 6a shows the computational time as a function of data cardinality where data dimensionality is fixed as 15 (i.e., we use (subsets) of the OCC-15 data set). Fig. 6b shows the computational time as a function of data dimensionality where data cardinality

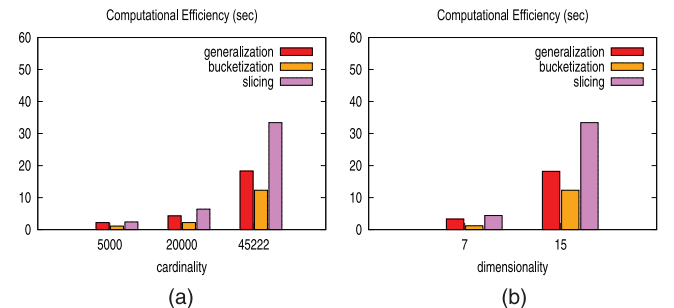


Fig. 6. Computational efficiency. (a) Cardinality and (b) dimensionality.

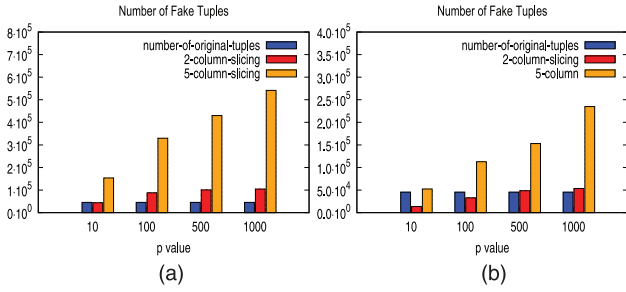


Fig. 7. Number of fake tuples: (a) OCC-7 and (b) OCC-15.

is fixed as 45,222 (i.e., all records are used). The results show that our slicing algorithm scales well with both data cardinality and data dimensionality.

### 6.3 Membership Disclosure Protection

In the second experiment, we evaluate the effectiveness of slicing in membership disclosure protection.

We first show that bucketization is vulnerable to membership disclosure. In both the OCC-7 data set and the OCC-15 data set, each combination of QI values occurs exactly once. This means that the adversary can determine the membership information of any individual by checking if the QI value appears in the bucketized data. If the QI value does not appear in the bucketized data, the individual is not in the original data. Otherwise, with high confidence, the individual is in the original data as no other individual has the same QI value.

We then show that slicing does prevent membership disclosure. We perform the following experiment. First, we partition attributes into  $c$  columns based on attribute correlations. We set  $c \in \{2, 5\}$ . In other words, we compare 2-column-slicing with 5-column-slicing. For example, when we set  $c = 5$ , we obtain five columns. In OCC-7,  $\{Age, Marriage, Gender\}$  is one column and each other attribute is in its own column. In OCC-15, the five columns are:  $\{Age, Workclass, Education, Education-Num, Capital-Gain, Hours, Salary\}$ ,  $\{Marriage, Occupation, Family, Gender\}$ ,  $\{Race, Country\}$ ,  $\{Final-Weight\}$ , and  $\{Capital-Loss\}$ .

Then, we randomly partition tuples into buckets of size  $p$  (the last bucket may have fewer than  $p$  tuples). As described in Section 5, we collect statistics about the following two measures in our experiments: 1) the number of fake tuples and 2) the number of matching buckets for original versus the number of matching buckets for fake tuples.

**The number of fake tuples.** Fig. 7 shows the experimental results on the number of fake tuples, with respect to the bucket size  $p$ . Our results show that the number of fake tuples is large enough to hide the original tuples. For example, for the OCC-7 data set, even for a small bucket size of 100 and only two columns, slicing introduces as many as 87,936 fake tuples, which is nearly twice the number of original tuples (45,222). When we increase the bucket size, the number of fake tuples becomes larger. This is consistent with our analysis that a bucket of size  $k$  can potentially match  $k^c - k$  fake tuples. In particular, when we increase the number of columns  $c$ , the number of fake tuples becomes exponentially larger. In almost all experiments, the number of fake tuples is larger than the number of original tuples. The existence of such a large number of fake tuples provides

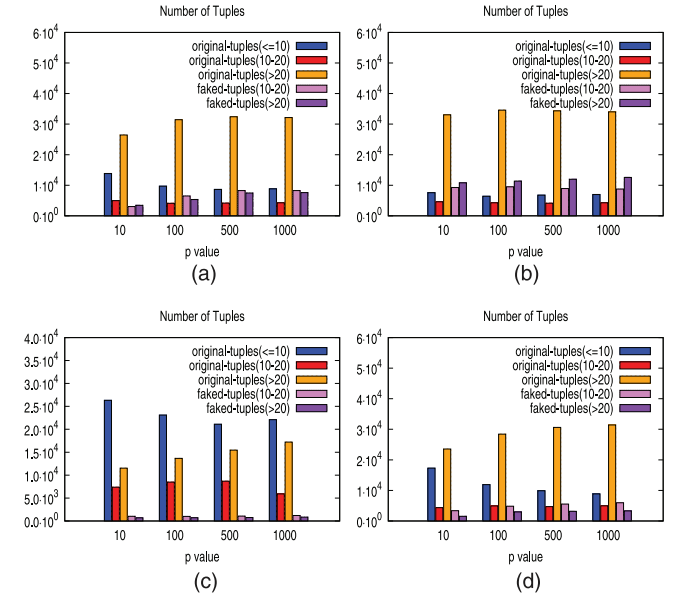


Fig. 8. Number of tuples that have matching buckets. (a) 2-column (OCC-7), (b) 5-column (OCC-7), (c) 2-column (OCC-15), and (d) 5-column (OCC-15).

protection for membership information of the original tuples.

**The number of matching buckets.** Fig. 8 shows the number of matching buckets for original tuples and fake tuples.

We categorize the tuples (both original tuples and fake tuples) into three categories: 1)  $\leq 10$ : tuples that have at most 10 matching buckets, 2) 10-20: tuples that have more than 10 matching buckets but at most 20 matching buckets, and 3)  $> 20$ : tuples that have more than 20 matching buckets. For example, the “original-tuples( $\leq 10$ )” bar gives the number of original tuples that have at most 10 matching buckets and the “fake-tuples( $> 20$ )” bar gives the number of fake tuples that have more than 20 matching buckets. Because the number of fake tuples that have at most 10 matching buckets is very large, we omit the “fake-tuples( $\leq 10$ )” bar from the figures to make the figures more readable.

Our results show that, even when we do random grouping, many fake tuples have a large number of matching buckets. For example, for the OCC-7 data set, for a small  $p = 100$  and  $c = 2$ , there are 5,325 fake tuples that have more than 20 matching buckets; the number is 31,452 for original tuples. The numbers are even closer for larger  $p$  and  $c$  values. This means that a larger bucket size and more columns provide better protection against membership disclosure.

Although many fake tuples have a large number of matching buckets, in general, original tuples have more matching buckets than fake tuples. As we can see from the figures, a large fraction of original tuples have more than 20 matching buckets while only a small fraction of fake tuples have more than 20 tuples. This is mainly due to the fact that we use random grouping in the experiments. The results of random grouping are that the number of fake tuples is very large but most fake tuples have very few matching buckets. When we aim at protecting membership information, we can design more effective grouping algorithms to ensure better protection against membership

disclosure. The design of tuple grouping algorithms is left to future work.

## 7 THE NETFLIX PRIZE DATA SET

We emphasized the applicability of slicing on high-dimensional transactional databases. In this section, we experimentally evaluate the performance of slicing on the Netflix Prize data set<sup>2</sup> that contains 100,480,507 ratings of 17,770 movies contributed by 480,189 Netflix subscribers. Each rating has the following format: (userID, movieID, rating, date), where rating is an integer in  $\{0, 1, 2, 3, 4, 5\}$  with 0 being the lowest rating and 5 being the highest rating.

To study the impact of the number of movies and the number of users on the performance, we choose a subset of Netflix Prize data as the training data and vary the number of movies and the number of users. Specifically, we choose the first  $n\text{Movies}$  movies, and from all users that have rated at least one of the  $n\text{Movies}$  movies, we randomly choose a fraction  $f\text{Users}$  of users. We evaluate the performance of our approach on this subset of Netflix Prize data.

**Methodology.** We use the standard SVD-based prediction method.<sup>3</sup> As in Netflix Prize, prediction accuracy is measured as the rooted-mean-square-error (RMSE).

We compare slicing against the baseline method. The baseline method will simply predict any user's rating on a movie as the average rating of that movie. Intuitively, the baseline method considers the following data publishing algorithm: the algorithm releases, for each movie, the average rating of that movie from all users. The baseline method only depends on the global statistics of the data set and does not assume any knowledge about any particular user.

To use slicing, we measure the correlation of two movies using the cosine similarity measure

$$\text{Sim}(m_1, m_2) = \frac{\sum \text{similarity}(m_{1i}, m_{2i})}{|\text{supp}(m_1) \cup \text{supp}(m_2)|},$$

$\text{similarity}(\text{rating}_1, \text{rating}_2)$  outputs 1 if both ratings are defined and they are the same; it outputs 0 otherwise.  $\text{supp}(m)$  is the number of users who have rated movie  $m$ . Then, we can apply our slicing algorithm to anonymize this data set.

Finally, since the data set does not have a separation of sensitive and nonsensitive movies, we randomly select  $\delta|n\text{Movies}|$  as sensitive movies. Ratings on all other movies are considered nonsensitive and are used as quasi-identifying attributes. Since the data set is sparse (a user has ratings for only a small fraction of movies), we "pad" the data set to reduce the sparsity. Specifically, if a user does not rate a movie, we replace this unknown rating by the average rating of that movie.

**Results.** In our experiment, we fix  $n\text{Movies} = 500$  and  $f\text{Users} = 20$  percent. We compare the RMSE errors from the original data, the baseline method, and slicing. For slicing,

2. The Netflix Prize data set is now available from the UCI Machine Learning Repository (<http://archive.ics.uci.edu/ml/datasets/Netflix+Prize>).

3. An implementation of the SVD method on the Netflix Prize data set is available here (<http://www.timelydevelopment.com/demos/NetflixPrize.aspx>).

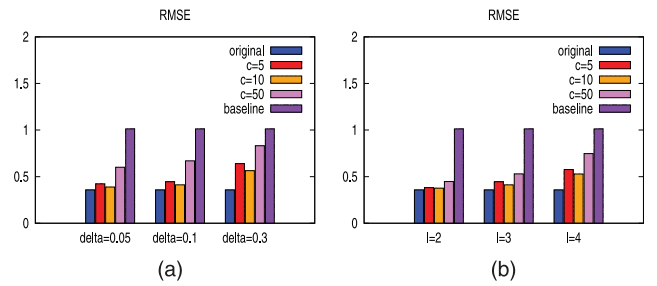


Fig. 9. RMSE comparisons: (a) RMSE versus  $\delta$  and (b) RMSE versus  $l$ .

we chose  $c \in \{5, 10, 50\}$ . We compare these five schemes in terms of RMSE. Fig. 9 shows the RMSE of the five schemes. The results demonstrate that we can actually build accurate and privacy-preserving statistical learning models using slicing. In Fig. 9a, we fix  $l = 3$  and vary  $\delta \in \{0.05, 0.1, 0.3\}$ . RMSE increases when  $\delta$  increases because it is more difficult to satisfy privacy for a larger  $\delta$  value. In Fig. 9b, we fix  $\delta = 0.1$  and vary  $l \in \{2, 3, 4\}$ . Similarly, RMSE increases when  $l$  increases.

The results also show the trade-off between the number of columns  $c$  and prediction accuracy RMSE. Specifically, when we increase  $c$ , we lose attribute correlations within columns as each column contains a small set of attributes. In the meanwhile, we have smaller bucket sizes and potentially preserve better correlations across columns. This trade-off is demonstrated in our results; we get the best result when  $c = 10$ . RMSE increases when  $c$  decreases to 5 or increases to 50. Therefore, there is an optimal value for  $c$  where we can best utilize this trade-off.

## 8 RELATED WORK

Two popular anonymization techniques are generalization and bucketization. Generalization [28], [30], [29] replaces a value with a "less-specific but semantically consistent" value. Three types of encoding schemes have been proposed for generalization: global recoding, regional recoding, and local recoding. Global recoding [18] has the property that multiple occurrences of the same value are always replaced by the same generalized value. Regional record [19] is also called multidimensional recoding (the Mondrian algorithm) which partitions the domain space into nonintersect regions and data points in the same region are represented by the region they are in. Local recoding [36] does not have the above constraints and allows different occurrences of the same value to be generalized differently. The main problems with generalization are: 1) it fails on high-dimensional data due to the curse of dimensionality [1] and 2) it causes too much information loss due to the uniform-distribution assumption [34].

Bucketization [34], [26], [17] first partitions tuples in the table into buckets and then separates the quasi identifiers with the sensitive attribute by randomly permuting the sensitive attribute values in each bucket. The anonymized data consist of a set of buckets with permuted sensitive attribute values. In particular, bucketization has been used for anonymizing high-dimensional data [12]. However, their approach assumes a clear separation between QIs and SAs. In addition, because the exact values of all QIs are released, membership information is disclosed. A detailed



comparison of slicing with generalization and bucketization is in Sections 2.2 and 2.3, respectively.

Slicing has some connections to marginal publication [16]; both of them release correlations among a subset of attributes. Slicing is quite different from marginal publication in a number of aspects. First, marginal publication can be viewed as a special case of slicing which does not have horizontal partitioning. Therefore, correlations among attributes in different columns are lost in marginal publication. By horizontal partitioning, attribute correlations between different columns (at the bucket level) are preserved. Marginal publication is similar to overlapping vertical partitioning, which is left as our future work (See Section 9). Second, the key idea of slicing is to preserve correlations between highly correlated attributes and to break correlations between uncorrelated attributes thus achieving both better utility and better privacy. Third, existing data analysis (e.g., query answering) methods can be easily used on the sliced data.

Recently, several approaches have been proposed to anonymize transactional databases. Terrovitis et al. [31] proposed the  $k^m$ -anonymity model which requires that, for any set of  $m$  or less items, the published database contains at least  $k$  transactions containing this set of items. This model aims at protecting the database against an adversary who has knowledge of at most  $m$  items in a specific transaction. There are several problems with the  $k^m$ -anonymity model: 1) it cannot prevent an adversary from learning additional items because all  $k$  records may have some other items in common; 2) the adversary may know the absence of an item and can potentially identify a particular transaction; and 3) it is difficult to set an appropriate  $m$  value. He and Naughton [13] used  $k$ -anonymity as the privacy model and developed a local recoding method for anonymizing transactional databases. The  $k$ -anonymity model also suffers from the first two problems above. Xu et al. [35] proposed an approach that combines  $k$ -anonymity and  $\ell$ -diversity but their approach considers a clear separation of the quasi identifiers and the sensitive attribute. On the contrary, slicing can be applied without such a separation.

Existing privacy measures for membership disclosure protection include differential privacy [6], [7], [9] and  $\delta$ -presence [27]. Differential privacy [6], [7], [9] has recently received much attention in data privacy. Most results on differential privacy are about answering statistical queries, rather than publishing microdata. A survey on these results can be found in [8]. On the other hand,  $\delta$ -presence [27] assumes that the published database is a sample of a large public database and the adversary has knowledge of this large database. The calculation of disclosure risk depends on the choice of this large database.

Finally, on attribute disclosure protection, a number of privacy models have been proposed, including  $\ell$ -diversity [25],  $(\alpha, k)$ -anonymity [33], and  $t$ -closeness [21]. A few others consider the adversary's background knowledge [26], [4], [22], [24]. Wong et al. [32] considered adversaries who have knowledge of the anonymization method.

## 9 DISCUSSIONS AND FUTURE WORK

This paper presents a new approach called slicing to privacy-preserving microdata publishing. Slicing overcomes the

limitations of generalization and bucketization and preserves better utility while protecting against privacy threats. We illustrate how to use slicing to prevent attribute disclosure and membership disclosure. Our experiments show that slicing preserves better data utility than generalization and is more effective than bucketization in workloads involving the sensitive attribute.

The general methodology proposed by this work is that: before anonymizing the data, one can analyze the data characteristics and use these characteristics in data anonymization. The rationale is that one can design better data anonymization techniques when we know the data better. In [22], [24], we show that attribute correlations can be used for privacy attacks.

This work motivates several directions for future research. First, in this paper, we consider slicing where each attribute is in exactly one column. An extension is the notion of *overlapping slicing*, which duplicates an attribute in more than one columns. This releases more attribute correlations. For example, in Table 1f, one could choose to include the *Disease* attribute also in the first column. That is, the two columns are  $\{Age, Sex, Disease\}$  and  $\{Zipcode, Disease\}$ . This could provide better data utility, but the privacy implications need to be carefully studied and understood. It is interesting to study the trade-off between privacy and utility [23].

Second, we plan to study membership disclosure protection in more details. Our experiments show that random grouping is not very effective. We plan to design more effective tuple grouping algorithms.

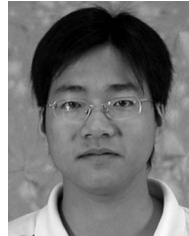
Third, slicing is a promising technique for handling high-dimensional data. By partitioning attributes into columns, we protect privacy by breaking the association of uncorrelated attributes and preserve data utility by preserving the association between highly correlated attributes. For example, slicing can be used for anonymizing transaction databases, which has been studied recently in [31], [35], [13].

Finally, while a number of anonymization techniques have been designed, it remains an open problem on how to use the anonymized data. In our experiments, we randomly generate the associations between column values of a bucket. This may lose data utility. Another direction is to design data mining tasks using the anonymized data [14] computed by various anonymization techniques.

## REFERENCES

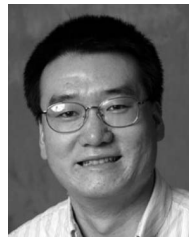
- [1] C. Aggarwal, "On  $k$ -Anonymity and the Curse of Dimensionality," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 901-909, 2005.
- [2] A. Blum, C. Dwork, F. McSherry, and K. Nissim, "Practical Privacy: The SULQ Framework," *Proc. ACM Symp. Principles of Database Systems (PODS)*, pp. 128-138, 2005.
- [3] J. Brickell and V. Shmatikov, "The Cost of Privacy: Destruction of Data-Mining Utility in Anonymized Data Publishing," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 70-78, 2008.
- [4] B.-C. Chen, K. LeFevre, and R. Ramakrishnan, "Privacy Skyline: Privacy with Multidimensional Adversarial Knowledge," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 770-781, 2007.
- [5] H. Cramt'er, *Mathematical Methods of Statistics*. Princeton Univ. Press, 1948.
- [6] I. Dinur and K. Nissim, "Revealing Information while Preserving Privacy," *Proc. ACM Symp. Principles of Database Systems (PODS)*, pp. 202-210, 2003.

- [7] C. Dwork, "Differential Privacy," *Proc. Int'l Colloquium Automata, Languages and Programming (ICALP)*, pp. 1-12, 2006.
- [8] C. Dwork, "Differential Privacy: A Survey of Results," *Proc. Fifth Int'l Conf. Theory and Applications of Models of Computation (TAMC)*, pp. 1-19, 2008.
- [9] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," *Proc. Theory of Cryptography Conf. (TCC)*, pp. 265-284, 2006.
- [10] J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209-226, 1977.
- [11] B.C.M. Fung, K. Wang, and P.S. Yu, "Top-Down Specialization for Information and Privacy Preservation," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 205-216, 2005.
- [12] G. Ghinita, Y. Tao, and P. Kalnis, "On the Anonymization of Sparse High-Dimensional Data," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE)*, pp. 715-724, 2008.
- [13] Y. He and J. Naughton, "Anonymization of Set-Valued Data via Top-Down, Local Generalization," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 934-945, 2009.
- [14] A. Inan, M. Kantarcioglu, and E. Bertino, "Using Anonymized Data for Classification," *Proc. IEEE 25th Int'l Conf. Data Eng. (ICDE)*, pp. 429-440, 2009.
- [15] L. Kaufman and P. Rousseeuw, "Finding Groups in Data: An Introduction to Cluster Analysis," John Wiley & Sons, 1990.
- [16] D. Kifer and J. Gehrke, "Injecting Utility into Anonymized Data Sets," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 217-228, 2006.
- [17] N. Koudas, D. Srivastava, T. Yu, and Q. Zhang, "Aggregate Query Answering on Anonymized Tables," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, pp. 116-125, 2007.
- [18] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Incognito: Efficient Full-Domain  $k$ -Anonymity," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 49-60, 2005.
- [19] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Mondrian Multi-dimensional  $k$ -Anonymity," *Proc. Int'l Conf. Data Eng. (ICDE)*, p. 25, 2006.
- [20] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Workload-Aware Anonymization," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 277-286, 2006.
- [21] N. Li, T. Li, and S. Venkatasubramanian, " $t$ -Closeness: Privacy Beyond  $k$ -Anonymity and  $\ell$ -Diversity," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, pp. 106-115, 2007.
- [22] T. Li and N. Li, "Injector: Mining Background Knowledge for Data Anonymization," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE)*, pp. 446-455, 2008.
- [23] T. Li and N. Li, "On the Tradeoff between Privacy and Utility in Data Publishing," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 517-526, 2009.
- [24] T. Li, N. Li, and J. Zhang, "Modeling and Integrating Background Knowledge in Data Anonymization," *Proc. IEEE 25th Int'l Conf. Data Eng. (ICDE)*, pp. 6-17, 2009.
- [25] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkatasubramanian, " $\ell$ -Diversity: Privacy Beyond  $k$ -Anonymity," *Proc. Int'l Conf. Data Eng. (ICDE)*, p. 24, 2006.
- [26] D.J. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J.Y. Halpern, "Worst-Case Background Knowledge for Privacy-Preserving Data Publishing," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, pp. 126-135, 2007.
- [27] M.E. Nergiz, M. Atzori, and C. Clifton, "Hiding the Presence of Individuals from Shared Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 665-676, 2007.
- [28] P. Samarati, "Protecting Respondent's Privacy in Microdata Release," *IEEE Trans. Knowledge and Data Eng.*, vol. 13, no. 6, pp. 1010-1027, Nov./Dec. 2001.
- [29] L. Sweeney, "Achieving  $k$ -Anonymity Privacy Protection Using Generalization and Suppression," *Int'l J. Uncertainty Fuzziness and Knowledge-Based Systems*, vol. 10, no. 6, pp. 571-588, 2002.
- [30] L. Sweeney, " $k$ -Anonymity: A Model for Protecting Privacy," *Int'l J. Uncertainty Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557-570, 2002.
- [31] M. Terrovitis, N. Mamoulis, and P. Kalnis, "Privacy-Preserving Anonymization of Set-Valued Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 115-125, 2008.
- [32] R.C.-W. Wong, A.W.-C. Fu, K. Wang, and J. Pei, "Minimality Attack in Privacy Preserving Data Publishing," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 543-554, 2007.
- [33] R.C.-W. Wong, J. Li, A.W.-C. Fu, and K. Wang, " $(\alpha, k)$ -Anonymity: An Enhanced  $k$ -Anonymity Model for Privacy Preserving Data Publishing," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 754-759, 2006.
- [34] X. Xiao and Y. Tao, "Anatomy: Simple and Effective Privacy Preservation," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 139-150, 2006.
- [35] Y. Xu, K. Wang, A.W.-C. Fu, and P.S. Yu, "Anonymizing Transaction Databases for Publication," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 767-775, 2008.
- [36] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A.W.-C. Fu, "Utility-Based Anonymization Using Local Recoding," *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 785-790, 2006.



on privacy preserving

**Tiancheng Li** received the BE degree in computer science from Zhejiang University in 2005. He is currently working toward the PhD degree in the Department of Computer Science and the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue University. His research interests are in the area of databases, data mining, information security and privacy, and applied cryptography, with a focus on privacy preserving data publishing and sharing.



**Ninghui Li** received the BE degree in computer science from the University of Science and Technology of China in 1993, and the MSc and PhD degrees in computer science from New York University, in 1998 and 2000. He is currently an assistant professor in Computer Science at Purdue University. Prior to joining Purdue University in 2003, he was a research associate at Stanford University Computer Science Department. His research interests are in security and privacy in information systems, with a focus on access control. He has worked on projects on trust management, automated trust negotiation, role-based access control, online privacy protection, privacy-preserving data publishing, and operating system access control. He has published more than 50 technical papers in refereed journals and conference proceedings and has served on the Program Committees of more than three dozen international conferences and workshops. He is a member of the ACM and the USENIX Association and a senior member of the IEEE.



**Jian Zhang** received the BS degree from Ocean University of China, the MS degree from Chinese Academy of Sciences, and the PhD degree from Carnegie Mellon University. He is an assistant professor in the Department of Statistics at Purdue University. His research interests include statistical machine learning, computational statistics, and information retrieval. He is a member of the IEEE and the IEEE Computer Society.



**Ian Molloy** is working toward the PhD degree in the Center for Education and Research in Information Assurance and Security at Purdue University. His research interests include access control, applied cryptography, and privacy. In particular, he is interested in the application of data mining and machine learning to problems in security and information assurance. His primary focus is on the application of data mining to problems in role engineering and role-based access control, and new models for access control and secure information sharing.

► For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).