Privacy Preserving in Knowledge Discovery and Data Publishing

B.Lakshmana Rao[1], G.V Konda Reddy[2], G.Yedukondalu[3]

***Abstract*— Knowledge Discovery is an indispensable technology for business and researches in many fields such as statistics, machine learning, pattern recognition, databases and high performance computing. In which Privacy Preserving Data Mining has the potential to increase the reach and benefits of data mining technology. This allows publishing a micro data without disclosing private information. Publishing data about individuals without revealing sensitive information about them is an important problem. K-anonymity and l-Diversity, generalization and Bucketization has been proposed as a mechanism for protecting privacy in micro data publishing.**

But k-anonymity and l-Diversity mechanisms are insufficient to protect the privacy issues like Homogeneity attack, Skewness Attack, Similarity attack and Background Knowledge Attack, 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.

A new privacy measure called slicing is proposed which is more flexible model for partitions the data both horizontally and vertically with 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 '-diversity requirement. Our experiments also demonstrate that slicing can be used to prevent membership disclosure

Keywords — Privacy preservation, data anonymization, data publishing, data security, data fusion, privacy.

I. INTRODUCTION

PRIVACY-PRESERVING publishing of micro data has been studied extensively in recent years. Micro data contains records each of which contains information about an individual entity, such as a person, a household, or an organization. Several micro data anonymization techniques have been proposed. The most popular ones are generalization [12], [15] for k-anonymity and Bucketization [14], [11], [7] for 'l-diversity. 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.

A. MOTIVATION OF SLICING

It has been shown [3], [5], [14] that generalization for k-anonymity losses 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 [14], [11], [7] has better data utility than generalization, it has several limitations.

First, Bucketization does not prevent membership disclosure [14]. 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 [15], 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) 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 permutated (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:

$t = \langle v_1, v_2 \ldots v_c \rangle$

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 ($1 <= i <= c$), 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.

II. SLICING

In this section, we present a novel technique called slicing for privacy-preserving data publishing with an example. Slicing has several advantages:

1) It preserves better data utility than generalization.
2) It preserves more attribute correlations with the SAs than bucketization.
3) It can also handle high-dimensional data and data without a clear separation of QIs and SAs.

We develop an efficient algorithm for computing the sliced table that satisfies '*l*-diversity. The 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.

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.

Example:
The following table shows an example micro data 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 is also partition tuples into buckets. Each bucket contains a subset of tuples. This horizontally partitions the 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 permutated 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 permutated and the values {(47906, dyspepsia), (47906, flu),(47905, flu), (47905, bronchitis)} are randomly permutated so that the linking between the two columns within one bucket is hidden.

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. SLICING ALGORITHMS:

We now present an efficient slicing algorithm to achieve '*l*-diverse slicing. Given a microdata table T and two parameters c and ', the algorithm computes the sliced table that consists of c columns and satisfies the privacy requirement of '*l*-diversity. Our algorithm consists of three phases:

1) Attribute partitioning
2) Column generalization
3) Tuple partitioning.

Attribute Partitioning: 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 attributes 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.

1) Measures of Correlation: Two widely used measures of association are Pearson correlation coefficient [2] and mean-square contingency coefficient [2]. 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 A1 and A2 with domains

$\{v_{11}; v_{12}; \ldots; v_{1d1}\}$ and
$\{v_{21}; v_{22}; \ldots; v_{2d2}\}$, respectively.

Their domain sizes are thus d1 and d2, respectively. The mean-square contingency coefficient between A1 and A2 is defined as

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

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 co occurrences of v_{1i} and v_{2j} in the data. Therefore, f_i and f_j are the marginal totals of

$$f_{ij}; f_{i\cdot} = \sum_{j=1}^{d_2} f_{ij} \text{ and } f_{\cdot j} = \sum_{i=1}^{d_1} f_{ij}.$$

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.

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) [6]. PAM starts by an arbitrary selection of k data points as the initial medoid. 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 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.

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 α. The parameter α determines the size of the sensitive column C_c, i.e., $|C_c| = \alpha$ If α=1, then $|C_c| = 1$, which means that $C_c = \{S\}$. And when c=2, slicing in this case becomes equivalent to bucketization. If α> 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 α 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 α-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.

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 [18], 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 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 2.1..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 [9]. The algorithms can be applied on the subtable containing only attributes in one column to ensure the anonymity requirement.

Tuple Partitioning: In the tuple partitioning phase, tuples are partitioned into buckets. We modify the Mondrian [9] 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). If the sliced table after the split satisfies '*l*-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 '*l* -diversity (line 5).

Algorithm tuple-partition(T, ℓ)
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

Algorithm diversity-check(T, T^*, ℓ)
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. $L[t] = L[t] \cup \{\langle p(t, B), D(t, B)\rangle\}$.
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

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 '*l*-diverse iff for all sensitive value s, p(t, s)<= 1 (lines 7 to 10). We now analyze the time complexity of the tuple partition algorithm. The time complexity of Mondrian [9] or kd-tree [1] 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 $O(n^2 \log n)$.

III. 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 set2 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, and 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 nMovies movies, and from all users that have rated at least one of the nMovies movies, we randomly choose fraction fUsers 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.3 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:

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

Similarity (rating1, rating2) outputs 1 if both ratings are defined and they are the same; it outputs 0 otherwise. 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 nMovies 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 nMovies=500 and fUsers=20 percent. We compare the RMSE errors from the original data, the baseline method, and slicing.

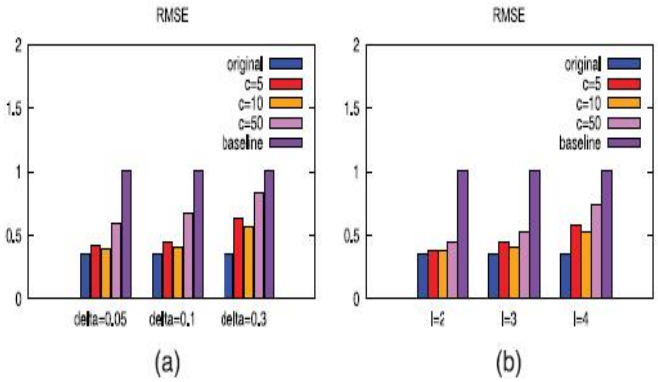

Fig. 3. RMSE comparisions: (a) RMSE versus _and (b) RMSE versus '.

For slicing, we chose c Є (5, 10, 50). We compare these five schemes in terms of RMSE. Fig. 3 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. 3a, we fix *l*=3 and vary δ Є (0:05, 0:1, 0:3). RMSE increases when δ increases because it is more difficult to satisfy privacy for a larger δ value. In Fig. 3b, we fix δ=0.1 and vary *l* Є (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.

4. DISCUSSIONS AND FUTURE WORK

This paper presents a new approach called slicing to privacy preserving micro data 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 [7], [10], 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 column. These 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, and Disease} and {Zip code, 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 [9]. Second, 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 [13], [1].

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 [4] computed by various anonymization techniques.

5. REFERENCES

[1] Machanavajjhala. A, Gehrke. J, Kifer. D and Venkitasubramaniam. M, "I-Diversity: Privacy Beyond k-Anonymity", Proc. Int'l Conl Data Eng. (ICDE), p. 24, 2006.

[2] 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.

[3] Agarwal. C and Agarwal. D, "On the Design and Quantification of Privacy Preserving Data Mining", Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 247-255, 2001.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9]K. LeFevre, D. DeWitt, and R. Ramakrishnan,"Mondrian Multidimensional
K-Anonymity," Proc. Int'l Conf. Data Eng. (ICDE), p. 25, 2006.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] X. Xiao and Y. Tao, "Anatomy: Simple and Effective Privacy Preservation," Proc. Int'l Conf. Very Large Data Bases (VLDB),
pp. 139-150, 2006.

B Lakshmana Rao: he received M.tech degree from JNTU Hyd,now working as a Asst.prof in the dept of CSE NBKR ISt,Vidyangar,Nellore