Leak-o-meter: Finding the Leak and Beyond

Data Plumbers*

1 Introduction

Sweeney demonstrated in [6] how data from different databases could be collated to find out what was meant to be private information. Miklau and Suciu [4] recently studied the following question in an information theoretic setting: given a view V that is made public, does V disclose any information about a query S the results of which one wants to keep a secret. The work provided an exact characterization as to when a view is secure with respect to a confidential query. But what maybe more desirable in a practical setting is a quantitative measure of *how much* information is disclosed as opposed to *whether or not* any information is disclosed. Miklau and Suciu also propose a measure of how much information does V "leak" with respect to S (we refer the reader to [4] for the terminology):

$$leak(S,V) = \sup_{s,v} \frac{Pr[s \subseteq S(I) | v \subseteq V(I)] - Pr[s \subseteq S(I)]}{Pr[s \subseteq S(I)]}. \quad (1)$$

1.1 Contributions

In this work, we have come up with a definition of leak for the case when one is given a view V, a query S and a database instance D. We also give an efficient algorithm based on Monte-Carlo sampling [2] that computes this leak for conjunctive queries (including joins).

2 Definition of Leak

Let us first examine the following variation of the leak in (1):

$$leak_1(S,V) = \sup_{s,v}(Pr[s \subseteq S(I) | v \subseteq V(I)] - Pr[s \subseteq S(I)]). \quad (2)$$

We note that this is the well known statistical distance [5] between the two distributions $S|V$ and S. We mention another interesting definition of leak in Section 6.

The problem we are tackling is different from the one considered in [4]. Miklau and Suciu defined leakage with respect to *any* view answer v while in our case the view answer is known: $v = V(D)$. Further, we just consider boolean queries[1], that is, the possible query answers are 0 and 1. Let p_{ub}

*Ankur Jain (ankur@cs.washington.edu) and Atri Rudra (atri@cs.washington.edu)

[1]This is WLOG as in the open world model, any query has an equivalent boolean query.

denote $Pr[S(I) = 1]$ and p_{vw} denote $Pr[S(I) = 1|V(I) = V(D)]$ where as in [4], the probabilities are over database instances I. It is easy to see that in this case the following definition is equivalent to (2):

$$leak(S, V, D) = p_{vw} - p_{ub} \tag{3}$$

3 Implementation

We first briefly mention the internal representation of a database in the tool. Note that in a schema $R(d_1, \cdots, d_m)$ where each domain d_i is of size s_i, the total number of possible tuples is $T = \prod_{i=1}^{m} s_i$ and thus, all possible (non-empty) database instances can be indexed by a T bit number (which ranges from 1 to $2^T - 1$). To complete the description of the encoding we need an ordering among the tuples and we use the most obvious one[2].

3.1 Exhaustive Search

As noted before, the basic task of the tool is to estimate the probabilities p_{ub} and p_{vw}. The first version of the tool tries to estimate these quantities by enumerating all possible database instances $\mathcal{D}$ and checking on how many of those the query evaluates to true (this estimates p_{ub}). To calculate p_{vw}, we do an extra check to see if the database instance satisfies the view (in the open world model), that is, we just count the number of database instances which yield the given view (denote this latter set by $\mathcal{D}_v$) and on which the query evaluates to true. The enumeration over all possible databases is done by defining an iterator which goes over the encodings of all instances in $\mathcal{D}$.

3.2 Monte Carlo Simulation

The exhaustive search method does not scale: we do not run into memory issues as we have a compact representation of the database instances but it would take for ever to finish on even small sized domains. The next version of the tool does sampling from both $\mathcal{D}$ and $\mathcal{D}_v$ and estimates p_{ub} and p_{vw}. Note that we are "sacrificing" accuracy for efficiency. We use a Monte Carlo simulation [2] where the database instances in $\mathcal{D}$ and $\mathcal{D}_v$ are assumed to be distributed uniformly. The first distribution is easy to generate: each tuple is assumed to be in a database instance with probability $\frac{1}{2}$. The uniform distribution on $\mathcal{D}_v$ is slightly more complicated because one has to ensure that it has some tuples which would yield the view: this is done using an idea similar to the indexing of database instances in the beginning of this section. We point out a small hack that we have used: if a "large" number of tuples yield a given tuple in the view then choosing each of them with probability $\frac{1}{2}$ would give an error[3] which would be smaller then the error of the Monte Carlo simulation.

[2] A tuple $(t_1, \cdots, t_m)$ has "value" $\sum_{i=1}^{m} t_i \prod_{j=1}^{i-1} s_i$ (we assume that any t_i is a value in $[0, s_i - 1]$). It is easy to see that each tuple has an unique value.

[3] Note that for the database instance to be in $\mathcal{D}_v$, atleast one of the tuples has to be in the database instance.

e	p
50	1
13	2
23	3
57	3
99	4

Figure 1: The Database instance D

#runs	leak	p_{vw}	time (in secs)
100	0.5200	1.0	0
1000	0.5190	1.0	6
5000	0.4998	1.0	20

Figure 2: Leak-o-meter on S_1

4 Results

In this section we will walk through three examples: one with large (or total) leak, one with partial leak and one with no leak. Consider the following toy example: There is a schema $Dir(e, p)$ where e is the employee id (this domain size is assumed to be 100) and p is the phone number (this domain size is 10). Now the company wants to publish the employee, phone pairs for a particular phone say 1, that is, $V(x, 1) : -Dir(x, 1)$. Figure 1 shows the database D.

We now consider the three queries $S_1() : -Dir(50, 1)$, $S_2() : -Dir(50, 9)$ and $S_3() : -Dir(50, 9), Dir(50, 1)$. Noting that we generate database instances using the uniform distribution, it is easy to see that the probabilities (p_{ub}, p_{vw}) for the three queries are $(\frac{1}{2}, 1)$, $(\frac{1}{2}, \frac{1}{2})$ and $(\frac{1}{4}, \frac{1}{2})$.

Figures 2, 3 and 4 show the results[4]from Leak-o-meter. Note that the exhaustive search would have to go through 2^{1000} database instances which would take forever while the Leak-o-meter on these queries takes atmost 36 seconds.

[4]#runs is the number of runs of the Monte Carlo Simulation and time is the time taken in seconds by the tool to spit the leak.

#runs	leak	p_{vw}	time (in secs)
100	0.0500	0.4700	0
1000	0.0004	0.4990	3
5000	0.0024	0.5012	34

Figure 3: Leak-o-meter on S_2

#runs	leak	p_{vw}	time (in secs)
100	0.2700	0.4700	1
1000	0.2590	0.4990	3
5000	0.2498	0.5012	36

Figure 4: Leak-o-meter on S_3

5 Current Status

To summarize, we have come up with a definition of leak for the case when one is given a view V, a query S and a database instance D. We also give an efficient algorithm based on Monte-Carlo sampling [2] that computes this leak for conjunctive queries (including joins).

Lastly, we point out implementation details which have not been covered till now. The tool supports chain join at the time of writing of this report: the extension to arbitrary joins is straight-forward and would be implemented soon. Another feature which needs to be looked into is the distribution of database instances from $\mathcal{D}_v$. Currently, for each tuple in the view, random database instances from $\mathcal{D}_v$ are generated as discussed in Section 3. If the view has multiple tuples then the union of the generated database instances are taken: after this the distribution does not remain uniform. We are investigating taking this skewed distribution into account and incorporating ideas from [2].

6 Future Work

There are two main ideas for future work. First note that while generating instances from $\mathcal{D}$, each tuple occurs with probability $\frac{1}{2}$. It is not unreasonable to assume that the adversary has some idea of c, the size of the database: for example, one can assume some upper limit on the distinct number of phone numbers in an organization instead of all possible seven digit numbers. In this case we can generate elements from $\mathcal{D}$ where each tuple appears with probability $\frac{c}{|\mathcal{D}|}$.

Another idea is to make use of the observation that elements of domains which do not appear as constants in the views and query behave similarly. For the ease of exposition, let us consider the simple example of a schema R with one column whose domain size is d. Assume that the view is $R(0)$ and the query is $R(1)$. Note that on any instance with the same number of elements from $\{2, c \ldots, d-1\}$ the query would evaluate to the same answer. Thus, we can collapse the set $\{2, \cdots, d-1\}$ to one "super-constant" s and treat all database instance which have the same number of occurrences of $0, 1$ and s as essentially the same. In other words, we have now reduced the space of $2^d - 1$ instance to one with $2^2(d-2)$ instances. Each element in this new space with i occurrences of s corresponds to $\binom{d-2}{i}$ instances in $\mathcal{D}$. Thus, we now associate weights of $\binom{d-2}{i}$ to each such "new" instances. The weights become more complicated for schemas with multiple columns and in the presence of join: they can be expressed in terms of product of suitable binomial coefficients. An interesting issue in the implementation of this idea is the fact that for even for moderately sized domains, these weights become large (for example $\binom{200}{100}$ cannot be represented as a `long` variable) and thus, we need support for large number arithmetic. Efficiently computing the binomial coefficient is an interesting problem in itself.

Miklau and Suciu cast polynomial identity testing [3] as testing the security of V with respect to S.

An alternate definition of the leak could be to measure the distance between the two polynomials [1]. It may however be noted that the polynomial testing problems are cast as decision problems and hence, the leakage tool would probably be a tester deciding if the leakage is within some threshold in this framework.

References

[1] N. Alon, T. Kaufman, M. Krivelevich, S. Litsyn, and D. Ron. Testing low-degree polynomials over $gf(2)$. In *RANDOM 2003*, 2003.

[2] R. M. Karp, M. Luby, and N. Madras. Monte carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[3] A. Klivans and D. Speilman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, 2001.

[4] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD 2004*, 2004.

[5] N. NIssan and D. Zuckerman. Randomness is linear in space. In *Proceedings of the ACM Symposium on Theory of Computing 1993*, 1993.

[6] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.