Information Protection in Dynamic Statistical Databases

Shiuh-Pyng Shieh and Chern-Tang Lin
Department of Computer Science and Information Engineering
National Chiao Tung University
Taiwan, ROC 30010
Tel: +886-3-5731876
Fax: +886-3-5724176
Email: ssp@csie.nctu.edu.tw

Abstract

A statistical database (SDB) is a database that contains sensitive records describing individuals but only statistical information is available. There are many inference control methods proposed to protect SDBs. In this article, we will briefly introduce three general approaches: conception, perturbation, and query restriction approaches. In addition, six criteria are also introduced to evaluate the features of these mechanisms. However, in practice, many SDBs are dynamically updated. This feature restricts the use of many inference control methods. Thus, in the rest of the article, we will present an efficient audit scheme, which is a query-restriction-based security-control method for protecting dynamic SDBs. This scheme guarantees the security of SDBs and needs less time and storage requirements than the traditional schemes while database systems are dynamically updated.

1. Introduction

A statistical database (SDB) is a database that contains sensitive records describing individuals but only statistical information is available. SDBs are mainly used for statistical analysis where only statistical queries, such as SUM, AVERAGE, COUNT are available and information of individuals cannot be disclosed. SDBs are used in many applications, such as census data, mortality data, and economic planning. A typical example of SDB is illustrated in Figure 1. In the SDB, the scores of individuals should not be disclosed, and therefore **AVERAGE**(ID = 1, Score), the average score of students with ID 1, is an illegal query. But statistical queries, such as **COUNT**(ALL) and **AVERAGE**(Address="New York", Score) are legal. Although users are only allowed to access the statistical information from an SDB, they can infer the confidential individual information by invoking a series of legal queries. When any confidential information is disclosed, the SDB is *compromised*. For example, both **AVERAGE**(Address = "New York", Score) and **AVERAGE**(Dept. =

"C.S.", Score) are legal queries. A user can infer the confidential information (the score of ID 3) by computing the difference between these two queries. If both queries are answered, the SDB will be compromised. Therefore, the SDB should deny one of the two queries to protect the individual information.

ID	Gender	Address	Dept.	Score
1	F	New York	C.S.	82
2	M	Washington	M.E.	75
3	M	Washington	C.S.	71
4	F	New York	C.S.	83

Figure 1. A statistical database

In practice, many SDBs are *dynamic*. That is, the individual records of an SDB need to be inserted, deleted and updated dynamically to keep statistical information fresh. A user may infer confidential information from the updates of a dynamic SDB. For example, when invoking the query **AVERAGE**(Gender = "M", Score) before and after inserting a new record with gender "M" into the SDB shown in Figure 1, the invoker can infer the new record's score from the change of the answers. Therefore, not only the old and the new values of an individual, but also the change of an SDB should be protected. Thus, the whole database needed be protected may be very huge. It will make the protection for dynamic SDBs more difficult than for static SDBs.

There are many inference control methods proposed to protect various database systems [1-3]. Those methods for SDBs can be classified into three classes: *conception*, *perturbation*, and *query restriction.* In this article, we will briefly introduce the three general approaches first. To evaluate the features of these mechanisms, six criteria [3] are also introduced. Then, in the following sections, we will describe some important works about the protection mechanism in SDB more clearly and discuss their features with the criteria. Finally, we will briefly present our scheme, a query-restriction-based security-control method for protecting dynamic SDBs [20]. This scheme guarantees the security of dynamic SDBs and needs less time and storage requirements than the traditional schemes.

2. Evaluation Criteria

Wortmann et al. [3] proposed six criteria to evaluate the security-control mechanisms for SDBs. These criteria include 1) security, 2) static/dynamic SDB, 3) richness, 4) bias, 5) consistency, and 6) cost.

1) Security

Of course, the security level that the protection methods can provide is the most

important point when we evaluate them. Some methods may be easy to implement but cannot guarantee absolute security, vice versa.

2) Static/Dynamic SDB

Suitability to dynamic SDB is also necessary. As the example mentioned in the previous section, the user can infer the confidential data by the reflection of dynamic operations. Hence, for a protection method, it ensures that any changes to the SDB are reflected in the statistics provided to users as soon as the changes have been taken place in the real world.

3) Richness

An ideal protection method should provide users with all relevant non-confidential information and at the same time protect all confidential information. That is, the method should not be under too much constraint for users. The user should get the information as many as possible under safe condition. Otherwise, it will lose the usefulness of the database.

4) Bias

Bias represents the difference between the unperturbed statistic and the expected value of its perturbed estimate. The bias should be zero or at least as small as possible.

5) Consistency

Consistency represents the lack of contradictions and paradoxes [Denn83]. Contradictions arise, for example, when repetitions of the same query yield different results, or the average statistic differs from the computed average using the sum and count statistics.

6) Cost

The cost of CPU time and storage space requirements must take into account. Whenever the system processes a query, the overhead occurs. In an on-line dynamic SDB environment, the processing cost is a significant factor. Implementation and education cost are also the factors besides the processing overhead.

3. Inference Control Methods for SDBs

To enhance the security of statistical databases, there are many inference control methods proposed. Those methods can be classified into three classes: *conception*, *perturbation*, and *query restriction.* In this chapter, we will illustrate some important works about the three approaches.

3.1 Conceptual Approach

The conceptual model provides a framework for investigating the security problem at the conceptual-data-model level [4]. A popular approach for the conceptual model is the *lattice model* [5, 6]. This model presents a framework for better understanding and investigating the security problem of SDBs, but gives too many constraints for users.

The lattice model describes SDB information in tabular form at different levels of aggregation. The interest in it stems from the fact that statistical information that is provided at different levels of aggregation may introduce redundant information. If confidential information is suppressed at the detailed level, such information might be disclosed due to more aggregate information.

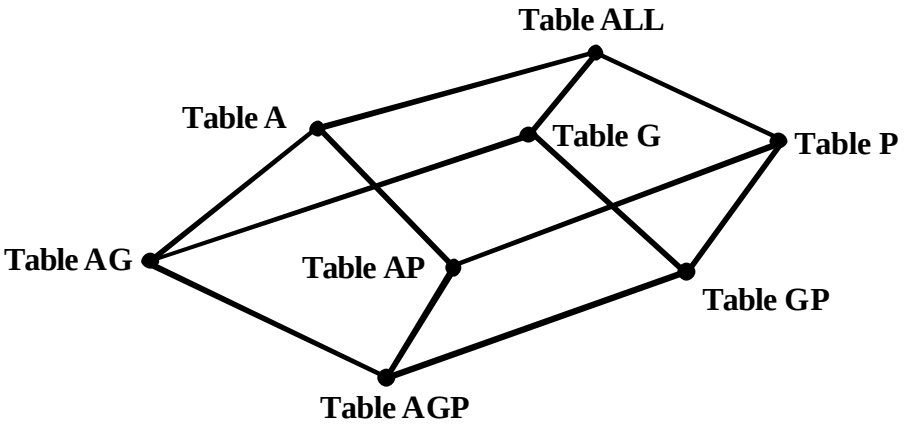


Figure 2 Lattice model

To illustrate, consider a college database with categorical attributes Age, Gender, and Position (A, G, P). The corresponding lattice model is shown in Figure 2. The most detailed way to represent this database in tabular form consists of a three-dimensional table AGP with dimensions A, G, and P (Figure 3).

Table AGP:

Position	Gender	Age			
		0-20	21-45	46-65	>65
Professor	M	0	1	9	0
	F	0	16	0	0
Vice Professor	M	1	20	48	0
	F	1	0	52	0
Others	M	24	2	9	49
	F	26	0	1	51

Table AG:

Gender	Age			
	0-20	21-45	46-65	>65
M	25	23	66	49
F	27	16	53	51

Table AP:

Position	Age			
	0-20	21-45	46-65	>65
Professor	0	17	9	0
Vice Professor	2	20	100	0
Others	50	2	10	100

Table GP:

Position	Gender	
	M	F
Professor	10	16
Vice Professor	69	53
Others	84	78

Table A:

Age			
0-20	21-45	46-65	>65
52	39	119	100

Table G:

Gender	
M	F
163	147

Table P:

Position		
Professor	Vice Professor	Others
26	122	162

Table ALL: 310

Figure 3. Lattice model

A particular elementary cell in this table might be, for example, the cell, where A=42, G=M and P=Professor. This tabular form can be aggregated into three

two-dimensional tables:

1. Table AG, where AGP is aggregated over the dimension P; an example of a cell in this case is the one in which A=42 and G=M.
2. Table AP, where AGP is aggregated over the dimension G; an example of a cell is the one in which A=42 and P=Professor.
3. Table GP, where AGP is aggregated over the dimension A; an example of a cell is the one in which G=M and P=Professor.

The aggregation performed in obtaining these three new tables is called microaggregation. The process can be repeated in order to obtain three one-dimensional tables:

1. Table A, where table AG is aggregated over the dimension G or table AP is aggregated over dimension P.
2. Table G, where table AG is aggregated over the dimension A or tables GP is aggregated over dimension P.
3. Table P, where table AP is aggregated over the dimension A or table GP is aggregated over dimension G.

The aggregation can be extended one step further, where a zero-dimensional table is obtained. This table contains only one cell, providing statistics for the database as a whole. Note that the set of all two-dimensional tables may sometimes disclose the elementary cell statistic of the three-dimensional table. The objective is to control the inference of sensitivity statistics. Here, we shall assume that a sensitive statistic one with a query set of size one. Thus, the male professor during 21-45 years old is sensitive.

The lattice model has proved a powerful and effective security model for studying the inference problem and proposed solutions. But it cannot provide the richness of databases and is not suitable for dynamic SDB.

3.2 Perturbation Approach

Perturbation approaches [7-13] introduce noise in the data, or perturb the answer to user queries while leaving the data in the SDB unchanged. They can be classified to *data perturbation* and *output perturbation*. Since these approaches cannot provide precise answers to users, bias and consistency are two major factors to evaluate perturbation methods. Here, we just give the basic idea of two approaches. The more details can see reference [7, 9, 10, 12].

3.2.1 Data Perturbation

Schlorer [12] suggested a data transformation scheme based on interchanging values in the records (called *swapping*). The objective is to swap enough values that nothing can be deduced from disclosure of individual records, but at the same time to preserve the accuracy of at least low-order statistics (A t-order statistic is some statistical quantity that can be computed from the values of exactly t attributes).

Schlorer defines a database $\boldsymbol{D}$ to be d-**transformable** if there exists at least one other database $\boldsymbol{D'}$ such that

$\boldsymbol{D}$ and $\boldsymbol{D'}$ have the same k-order frequency counts for k=0,1... d, and

$\boldsymbol{D}$ and $\boldsymbol{D'}$ have no records in common.

Due to the computational requirement of the method, it is only feasible to consider it for static SDBs. And there is a need for a one-to-one mapping between the original database and the perturbed database. Although several alternative methods discussed [11], further investigation is required. And as shown in [11], the method may in some cases have an error of up to 50%.

In general, data swapping has not been developed enough to be seriously considered for SDBs.

3.2.2 Output Perturbation

Denning [7] proposed a method that is comparable to ordinary random sampling where a sample is drawn from the query set itself (called *Random-Sample Queries*). Given a characteristic formula $\boldsymbol{C}$, a set of records that satisfies $\boldsymbol{C}$ is determined. For each record r in the query set, the system applies a Boolean formula $f(\boldsymbol{C}, r)$ to determine whether this record is to be included in the sampled query set. The function f is designed in such a way that there is a probability P that the record is included in the sampled query set. The probability P can be set by database administrators. The required statistics are computed based on the sampled query set. The statistics computed from the sampled query set have to be divided by P in order to provide a corresponding unbiased estimator. For example, if the response to a count query based on the sampled query set is n^*, an estimator for the true count is n^*/P.

The probability of a record being included in the query set P is either fixed or variable. When P is fixed there should be a query-set-size restriction if P is large. Otherwise, there is a high probability of including all the records of a small query set, thus compromising the database. When P is variable, it should approach 0 for a query -set-size approaching 1 in order to avoid compromising the database. However, this will be high inconsistent.

3.3 Query Restriction Approach

Query restriction methods impose extra restriction on queries which includes restricting the query set size [14], controlling the overlap among successive queries [15], auditing [16], partitioning [17, 18], and suppressing cells [19]. Some of them cannot guarantee high security assurance, while others limit the usefulness of the SDBs. The following subsections will discus these five methods and give some comments with respect to the criteria mentioned in section 2.

3.3.1 Query-Set-Size Control

For each query, the query-set-size control method restricts the number of individuals, $|C|$. The database system responds the answer of the query only if $|C|$ satisfies the condition [14]

$$K \le |C| \le L - K$$

where L is the size of the database (the number of individuals represented in the database) and K is a parameter set by the database administrator. Obviously, K should satisfy the condition

$$0 \le K \le L/2,$$

otherwise no query can be answered.

Summarily, the main advantage of query-set-size control method is easily implemented so that this scheme is suitable for dynamic SDBs. In [21], however, it was shown that by using a snooping tool called "*tracker*" it was possible to compromise the SDB that was protected with the query-set-size control scheme alone.

3.3.2 Query-Set-Overlap Control

Dobkin et al. [15] pointed out that it is possible for a user to determine the value of a particular individual if two queries *overlap* greatly. (The overlapping part of two queries is the set of individuals included in both queries.) They investigated the correlation between the overlapping size (r) of queries and the number of queries (S) that suffice to compromise the database. A lower bound on S was deduced as

$$S \ge 1 + (k - (l + 1)) / r,$$

where k denotes the query size (the number of individuals included in the query) and l denotes the number of individuals whose values have been known by users. Thus, for a secure SDB, l should be equal to zero, and the minimum number of queries (S) for compromising the SDB should be larger that $1+(k+1)/r$. Following this deduction, the authors of [15] pointed out that there do exist mechanisms (bounding the overlap and the number of queries) that can protect a database.

However, query-set-overlap control method suffers from the drawback such as that the control method will seriously impair the richness of the database. For example, if the individuals included in a new query are the subset of a previous query, the query-set-overlap control method will deny the new query since both queries overlap greatly (even that the new query may not cause the compromise of SDBs). Because of the lack of practical interest, the query-set-overlap control method is also unsuitable for dynamic SDBs.

3.3.3 Auditing

Auditing of an SDB involves keeping up-to-date logs of all queries made by each user and constantly checking for possible compromise whenever a new query is issued [16]. Auditing has the advantages such as allowing the SDB to provide users with unperturbed response that will not make compromised. Besides, auditing mechanism can avoid linear system attacks [14]. For example,

$$x_1 + x_2 + x_3 = q_1$$

$$x_1 + x_2 + x_4 = q_2$$

$$x_1 + x_3 + x_4 = q_3$$

$$x_2 + x_3 + x_4 = q_4$$

The value x_1 can be compromised by computing

$$x_1 = 1/3\ (q_1 + q_2 + q_3 - 2q_4).$$

In fact, almost all of compromised conditions occurring in SDBs due to linear system attack.

One of the major drawbacks of auditing, however, is its excessive CPU time and storage space requirement to store and process the accumulated logs.

A practical approach based on auditing was proposed by Chin and Ozsoyoglu[16]. The approach maintains a matrix to audit the history of user's queries and detects all of the possible compromised conditions from linear system attack.

In Chin's scheme, the SDB consists of n individuals x_i, $1 \leqq i \leqq n$. For notational simplicity, each individual x_i is assumed to have a single protected numerical attribute value, and each answered query reveals a set of individual records $\{x_i, x_j, x_k, ...\}$. Hence, each answered query can be represented by a vector $(a_1, a_2, ..., a_n)$, where $a_i = 1$, if x_i is accessed in this query. The user's knowledge space KS is the vector space spanned by the set of vectors of answered queries AQ. Formally, KS has the following properties.

1) If $\bar{q} \in AQ$, then $\bar{q} \in KS$.

2) If $\bar{q} \in KS$, then $b * \bar{q} \in KS$; b is a real number.

3) If $\bar{q_1}, \bar{q_2} \in KS$, then $\bar{q_1} + \bar{q_2} \in KS$.

4) Nothing else is in *KS*.

KS can be represented by a maximal set of non-redundant vectors of *AQ*. For example in Figure 1,

$\bar{q_1}$ =(1,0,0,1) (SUM of the scores of the people living in New York)

$\bar{q_2}$ =(1,0,1,1) (SUM of the scores of the people majoring in C.S.).

We have

$$KS = \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

where c_i represents the column associated with individuals x_i. Notice that the vectors in *KS* are linear independent. Therefore, the number of rows cannot exceed the number of columns in *KS*. The SDB is compromised if there exists a vector of the form (0, ...,0,1,0, ...,0) in *KS*.

Unfortunately, Chin's scheme suffers from space explosion problems if the SDB is dynamically updated. In Chin's scheme, when an individual of an SDB is inserted, a new column corresponding to this individual is inserted to *KS*. Since the new individual has not yet been queried, all entries of the new column are zeros. On the other hand, when an individual is deleted, the corresponding column, called the *dangling column*, cannot be directly removed from the *KS* matrix for the protection of individual information.

If we directly delete the dangling columns to reduce the size of *KS*, the deletion may cause both false alarms and security disclosure. A false alarm is raised when a vector with a single "1" is found in the audit matrix but the corresponding individual is not disclosed. On the other hand, security disclosure occurs when the audit matrix does not have any vector with a single "1", but the secret of an individual is disclosed. For example in Figure 4, the individual x_2 is deleted from SDB. If we remove the corresponding column c_2 in *KS*, the audit matrix will report that x_4 is disclosed and

$$\boldsymbol{KS} = \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

x_2 is deleted from the SDB and the c_2 column is removed →

$$\boldsymbol{KS} = \begin{matrix} c_1 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

the SDB is compromised. (That is, according to the second row recorded in new *KS*, the vector (0, 0, 1, 0, 0, 0) contains a single "1" at the position of x_4.) In fact, x_4 is still undisclosed at this time. Thus, a false alarm has been raised.

Another example illustrated for security disclosure is shown in Figure 5. In this example, the individual x_1 is deleted from the SDB. It seems reasonable to delete the column c_1. However, the deletion of the column will cause disclosure of secret information. Assume that a new answered query, (0,1,0,1,1,1), is invoked in *KS* after the deletion. The audit scheme will check *KS* and consider it as a redundant answerable query, which is the same as r_1. As a result, *KS* remains unchanged and the query is answered. Thus, the secret information of the deleted x_1 is disclosed.

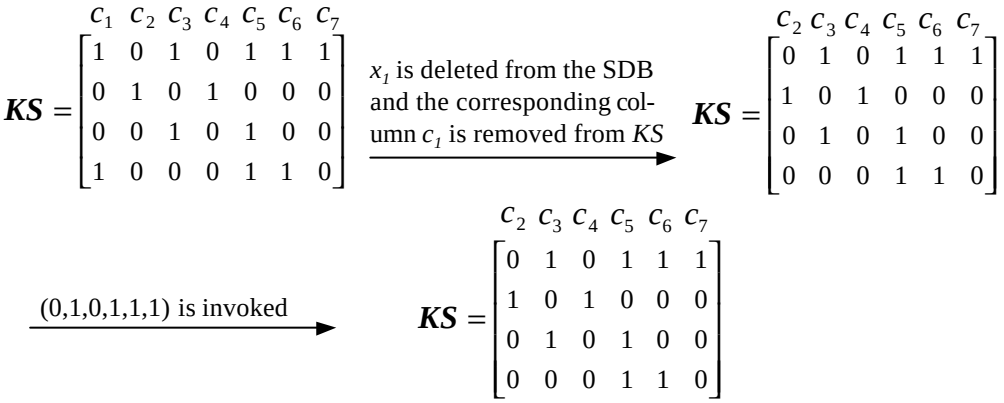


Figure 5. Deletion that causes disclosure of secret information.

The two examples above demonstrate that we cannot arbitrarily remove a column in a *KS* when the corresponding individual is deleted from the SDB. Therefore, the size of *KS* will only be expanded without any upper bound when the individuals of a finite-size SDB are dynamically inserted, deleted or updated. It is possible to have a large *KS* for a small SDB. Substantial memory and CPU time are wasted in handling these columns. It is not efficient to check the entire *KS* matrix for every query, when the number of the rows and the columns in *KS* is large. To cope with the problem, Chin imposes the restriction on the scheme that it can only be used in static SDBs. The restriction limits the use of the scheme. A method for the reduction of *KS* is desirable.

In [20], we propose an algorithm for the reduction of KS size. With this algorithm, Chin's scheme can be enhanced so that it can be used in a dynamic SDB. As described above, in order to guarantee the security of an SDB, all dangling columns cannot be arbitrarily deleted from the *KS*. However, it is possible to delete part of the dangling columns if the deletion will not cause the false alarm or the disclosure of any individual information. We call the removable part of audit matrix is a *related*

garbage set.

In an audit matrix, an entry can only be either '1' or '0'. A column and a row are *directly related* if their shared entry is '1'. Indirectly related relation can be defined recursively. A column/row is *indirectly related* to a column/row if a directly related column/row of the former is directly/indirectly related to the latter. If a column/row is directly or indirectly related to another column/row, then they are *related*. Otherwise, they are *unrelated*. All related columns and rows form a *related set*. All elements of a related set are related to each other, and no element outside of the related set can be related to any element of the set. For example, in Figure 6, r_1 and r_4 are directly related to c_1; r_1 are indirectly related to r_4; $\{c_1, c_3, c_5, c_6, c_7, r_1, r_3, r_4\}$ is a related set.

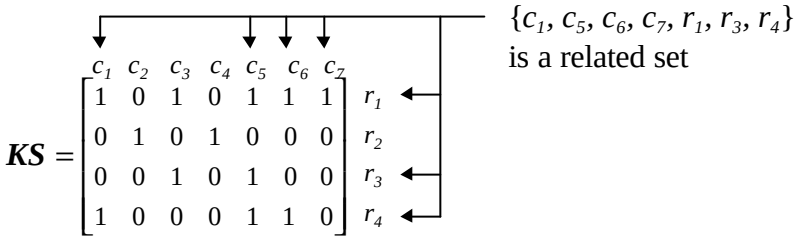


Figure 6. A related set

Definition 1: Let $c_1, c_2, ..., c_k, r_1, r_2, ..., r_l$ represent all elements of a related set in the audit matrix. If $c_1, c_2, ..., c_k$ are dangling columns, then
(1) $c_1, c_2, ..., c_k$ and $r_1, r_2, ..., r_l$ are *garbage columns* and *rows*, respectively, and
(2) the related set is called a *related garbage set*.

Since the garbage columns and rows of a related set are unrelated to other columns and rows, they can be removed without affecting the subsequent security analysis of the audit matrix. In [20], we have proposed an algorithm FINDING_GARBAGE based on the concept that garbage columns and rows are related. Whenever an individual is deleted, the algorithm is able to find all the columns and rows related the new dangling column. If these columns are also dangling, then these columns and rows are all garbage and can be removed.

Although we can use this method to reduce the memory requirement and improve the performance of Chin's audit scheme, FINDING_GARBAGE itself also introduce overhead for the deletion of individuals from an SDB. In section 4, we will introduce a new scheme to construct the knowledge space of the SDB. It uses less space to maintain the audit matrix, and its garbage information in the knowledge space can be easily found and removed without the need of invoking the FINDING_GARBAGE procedure.

3.3.4 Partitioning

The basic idea of partitioning is to cluster individual entities in a number of mutually exclusive subsets, called atomic populations [17]. The statistical properties of these atomic populations constitute the raw materials available to the database users. As long as atomic populations do not contain precisely one individual record, a high security level can be attained.

Schlorer [12] has investigated a large number of practical databases and found that a considerable number of atomic populations with only one entity will emerge. Clustering such populations with larger ones leads to serious information loss.

In order to cope with the problem of atomic populations of size 1, it was proposed in Chin and Ozsoyoglu [17], to add dummy records to the database. However, bias will be suffered from it.

3.3.5 Cell Suppression

Cell suppression [19] is one of the techniques typically used by census bureaus for data published in tabular form. Cell suppression has been investigated for static SDBs. The basic idea is to hide the cells that may cause confidential information to be disclosed. Other cells of non-confidential information which may lead to a disclosure of some confidential information also have to be suppressed (this is called complementary suppression).

Cox [19] has studied the determination of complementary suppressed cells. He shows that the determination of a minimum set of complementary suppression involves a great deal of computational complexity. So cell suppression is limited by the computational complexity of the analysis procedure of determining the complementary suppressed cells. And the set of all possible statistics for a database with **N** fields of attributes in each record corresponds to an **N**-dimensional table. When **N** is large, applying cell suppression to a table of this size may not be tractable.

In [6], they show that cell suppression becomes impractical if an arbitrary complex syntax for queries is allowed. (With such syntax, suppression of complete tables from the lattice model might be necessary.)

4. Audit Scheme for Dynamic SDBs

Although the traditional audit approach mentioned in section 3 can guarantee the security of SDBs and provide precise responses, it is impractical for dynamic SDBs since the CPU time and storage space explosion problem. In this section, we will introduce an audit scheme which needs less storage space for auditing users' queries.

Thus the CPU time of analyzing the security of SDBs is also reduced [20]. Since the cost is significantly reduced, the audit scheme will be suitable for dynamic SDBs.

In the statistical queries of SDBs, individuals with the same characteristics tend to be queried together, and individuals with different characteristics tend not to be queried together. It is possible to speed up the security analysis process by taking advantage of the characteristics.

Let G_j represent the set of individuals that were always queried together. The knowledge space in our scheme is represented as a set of vector spaces, VS_1, VS_2,...,VS_m, and an untouched set Z of never accessed individuals. VS_i provides the knowledge regarding to the individuals that were accessed at least in a query. VS_i is represented in the matrix form where the columns are associated with the groups, the rows are linearly independent answered query vectors, and its entry indicates the status of the groups. A '1' entry indicates that all individuals of a group are accessed. A '0' entry indicates that all individuals of a group are not accessed.

There are three operations that are used to reconstruct the *VS* set: creating new *VSs* and new groups, splitting groups, and merging independent *VSs* into a new one. We will discuss them in the following.

4.1 The Operations of the Audit Scheme

Creating New *VSs* and New Groups

If some individuals in Z are queried by a new answered query, they will form a new group. If all the queried individuals belong to Z, a new vector space VS_i is created which only contains a single column associated with the new group since the new group is never queried together with other groups. If only a subset of the queried individuals belongs to Z, a new column associated with the subset will be added to the VS_i whose groups are also queried in the new answered query. As an example, assume that initially the VS set are empty, and the set Z contains all individuals x_1, x_2,...,and x_7. When the first answered vector is invoked, the individuals which are accessed in this query should be grouped together and the others should remain in the Z set. If the first vector is $\overline{q_1}$ = (1,0,1,0,1,1,1), then

$\boldsymbol{VS_1} = \overset{G_1}{[1]}$, where $G_1 = \{x_1, x_3, x_5, x_6, x_7\}$ and $Z = \{x_2, x_4\}$.

Assume the second vector $\overline{q_2}$ = (0,1,0,1,0,0,0) is invoked. The accessed individuals

in $\overline{q_1}$ and $\overline{q_2}$ are totally unrelated, and therefore a new vector space VS_2 and a new group G_2 are created. At the same time, the Z set must also be changed. Therefore, the *VSs* become

$\boldsymbol{VS_1} = \overset{G_1}{[1]}$ and $\boldsymbol{VS_2} = \overset{G_2}{[1]}$, where $G_1 = \{x_1, x_3, x_5, x_6, x_7\}$, $G_2 = \{x_2, x_4\}$, and $Z = f$.

Splitting Groups

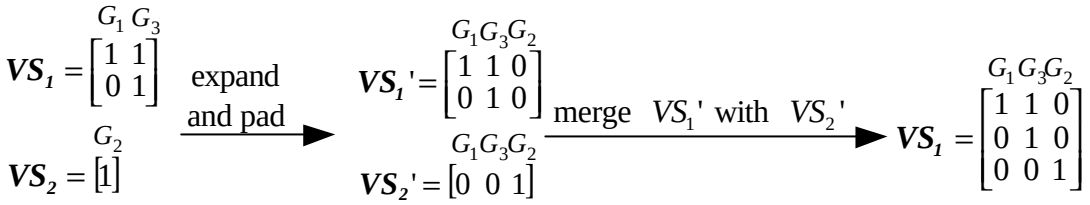
When individuals that have been always queried together and included in the same group are not queried together in the new answered query, the group must be split. Because there are only two possible values, 0 or 1, in an answered vector, the group must be split into two new groups: one group associated with the 1's and the other group associated with the 0's in the new answered vector. The two new columns associated with the two new groups have the same values as the old column associated with the original group. A new row will also be inserted into the new vector space VS_i. With the same example above, assume that the third answered query is $\overline{q_3}$ = (0,0,1,0,1,0,0), where only x_3 and x_5 are queried together. Thus, G_1 is split into two new groups, G_1 and G_3. The new VS_1 and groups are listed as follows:

$$\boldsymbol{VS_1} = \begin{bmatrix} \overset{G_1}{1} & \overset{G_3}{1} \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{VS_2} = \overset{G_2}{[1]}, \text{ where } G_1 = \{x_1, x_6, x_7\},\ G_2 = \{x_2, x_4\},\ G_3 = \{x_3, x_5\}, \text{ and } Z = f .$$

Notice that, except the second row in the new VS_1, the two new columns associated with the new G_1 and G_3 have the same values as the old one associated with the old G_1.

Merging the *VSs*

When the individuals of different *VSs* are queried together, these *VSs* must be merged into a new one. Thus, a $h \times m$ VS_i and a $k \times n$ VS_j will be merged into a $(k+h) \times (n+m)$ VS_k. The $k \times n$ VS_j must be expanded by padding with m all-'0' columns before merging with a $h \times m$ VS_i. Similarly, the $h \times m$ VS_i also need to be expanded by padding with n all-'0' columns. Using the previous example, assume that the fourth query $\overline{q_4}$ =(0,1,1,1,1,0,0) is invoked, then VS_1 and VS_2 are merged because that G_2 and G_3 are queried together. Note that the new query vector will not be inserted into the new VS_1 because it can be computed as r_2+r_3, and thus is not linearly independent of the rows of VS_1. The merging process is shown as follows:



The *VS* set is reconstructed if any of the three operations described above is invoked by a new answerable query. The reconstruction of the *VS* set is scarcely needed if individuals with similar characteristics tend to be queried together, and individuals with the different characteristics tend not to be queried together in the answered queries. In this case, the time spent on reconstructing the VSs can be ignored.

With the *VS* set presented in our scheme, we are able to distinguish illegal queries. The checking process within a VS is similar to that within a KS in Chin's scheme. An SDB is compromised if there exists a row containing a single '1'-entry in its *VSs* and the corresponding group contains only a single individual. Otherwise, the SDB is still secure after answering the query.

4.2 Updates in A Dynamic SDB

The insertion and deletion of individuals in our scheme is easy. In a dynamic SDB, whenever a new individual is inserted into the SDB, the individual is directly inserted into the set *Z* because it is never accessed before. On the other hand, when an individual is deleted from the database, it can be removed from *Z* without modifying *VSs* if it belongs to the set *Z*. Otherwise, we must consider two cases. Assume that the deleted individual x_i belongs to the group G_j which is contained in VS_k. In the first case, G_j contains at least one individual, excluding x_i, that has not yet been deleted. All we have to do is to mark x_i as deleted. In the second case that all individuals, except x_i, contained in G_j have been marked as deleted, G_j must be marked as deleted. If a group is marked as deleted, the system herein check whether all groups in the same *VS* are deleted. If all groups of this VS are deleted, then the *VS* and its groups can be removed. Since the check is very simple, the cost will be limited and ignored. (Refer to [20] for the details of deletion.)

In the traditional audit approach, e.g. Chin's scheme, it takes no more than $O(KN)$ steps to check the security of the SDB and determine whether a new query vector $\bar{q} \in K \times N$ *KS* [13], where *N* is equal to the sum of the total number of individuals in an SDB (n_a) and that of the deleted ones (n_d). In our scheme, consider an average case where each group contains *u* individuals and each vector space contains equal number of columns (*n*) and rows (*k*). The complexity of our scheme for checking the security

of the SDB becomes O($k \times n$), which can also be represented as O($\frac{K}{v} \times \frac{N}{uv}$). Furthermore, in a dynamic SDB where ninety percent of the individuals ($n_d/N = 90\%$) are deleted and their corresponding columns are garbage, the complexity can be further reduced to O($\frac{K}{10v} \times \frac{N}{10uv}$). Comparing with the O($K \times N$) of Chin's scheme, our scheme performs better.

Reference

[1] M. Mogenstern, "Controlling logical inference in multilevel database systems," *Proc. IEEE CS Symp. Security and Privacy*, pp.245-255 (Apr. 1988).

[2] H. S. Delugach and T. H. Hinke, "Wizard: A database inference analysis and detection system," *IEEE Tran. on Knowledge and Data Engineering*, Vol.8(1), pp.56-66 (Feb. 1996).

[3] J. C. Wortmann and N. R. Adam, "Security-Control methods for statistics databases: A comparative study," *ACM Computing Surveys,* Vol. 21(4) pp. 515-554 (Dec. 1989)

[4] F. Y. Chin and G. Ozsoyoglu, "Statistical database design," *ACM Trans. on Database Syst.* Vol. 6(1) pp. 113-139 (Mar. 1981).

[5] D. E. Denning, "A security model for the statistical database problem," In *Proceedings of the 2nd International Workshop on Management,* pp. 1-16 (1983).

[6] D. E. Denning and J. Schlorer, "Inference control for statistical databases," *Computer,* Vol. 16(7) pp. 69-82 (July 1983).

[7] D. E. Denning, "Secure statistical databases under random sample queries," *ACM Trans. on Database Syst.* Vol. 5(3) pp. 291-315 (Sept. 1980)

[8] G. Ozsoyoglu and T. A. Su, "On inference control in semantic data models for statistical databases," *Journal of Computer and System Sciences*, Vol.40(3), pp.405-443 (Jun. 1990).

[9] S. B. Reiss, "The practicality of data swapping," *Technical Report No. CS-48*, Dept. of Computer Science, Brown Univ., Providence, R.I. (1979).

[10] S. B. Reiss, "Practical data-swapping: The first steps," In *Proceedings 1980 Symp. on Security and Privacy, IEEE Computer Society* pp. 38-45 (Apr. 1980).

[11] S. B. Reiss, "Practical data-swapping: The first steps," *ACM Trans. on Database Syst.* pp. 20-37 (Mar. 1984).

[12] J. Schlorer, "Security of statistical databases: Multidimensional transformation," *ACM Trans. on Database Syst.* Vol. 6(1) pp. 95-112 (Mar. 1981).

[13] J. F. Traub, Y. Yemini and H. Wozniakowski, "The Statistical Security of a Statistical Database," *ACM Trans. on Database Syst*, Vol. 9(4) pp.672 - 679 (Dec. 1984).

[14] D. E. Denning, "Cryptography and Data Security," Addison-Wesley, Reading Mass.

[15] D. Dobkin, A. K. Jones and R. J. Lipton, "Secure databases: Protection Against User Inference," *ACM Trans. on Database Syst.* Vol. 4(1) pp. 97-106 (Mar. 1979).

[16] F. Y. Chin and G. Ozsoyoglu, "Auditing and inference control in statistical databases," *IEEE Trans. on Softw. Eng.* pp. 574-582 (Apr. 1982).

[17] F. Y. Chin and G. Ozsoyoglu, "Security in partitioned dynamic statistical databases," In *Proceedings of the IEEE COMPSAC,* pp. 594-601 (1979)

[18] M. McLeish, "Further result on the security of partitioned dynamic statistical databases," *ACM Trans. on Database Systems*, Vol.14(1), pp.98-113 (Mar. 1989).

[19] L. H. Cox, "Suppression methodology and statistical disclosure control," *J. Am. Stat. Assoc.* Vol. 75(370) pp. 377-385 (June 1980).

[20] Shiuh-Pyng Shieh and Chern-Tang Lin, "Auditing user queries in dynamic statistical databases," Information Science, Vol. 113(1-2), pp. 131-146 (Jan. 1999).

[21] Denning, D. E., Denning, P. J. and Schwartz, M. D., "The tracker: A threat to statistical database security," *ACM Trans. on Database Syst.* Vol. 4(1) pp. 76-96 (Mar. 1979).