The Fingerprinted Database

Neal R. Wagner[1] **Robert L. Fountain** **Robert J. Hazy**

The University of Texas at San Antonio
Division of Mathematics, Computer Science and Statistics
San Antonio, Texas 78285

AT&T Material Management Services
5 Woodhollow Road
Parsippany, New Jersey 07054

Abstract. A fingerprinting Database Management System presents a unique version or view of the data to each user. The view contains small pseudo-random modifications to selected data items. The objective of this system is to identify users who divulge proprietary information which they have obtained through legitimate access. Even if the user makes further modifications to the data before its misuse, a statistical procedure will identify the source of misuse with any desired degree of certainty.

1. Introduction

This article presents an approach to data security which identifies the source of misused or leaked data by inserting small, user-specific modifications to the data itself. These modification (called *fingerprints*) uniquely identify the particular person (called the *opponent*) making unauthorized use of the data. This article proposes distribution of a distinct *fingerprinted view* of the database to each user.

There have been many speculative accounts of fingerprinting schemes, including material in an earlier article by the first author [Wagn 83] and even a method described in a popular novel [Clan 87, p. 87]. Other examples range from the "dirty, strange" fingerprints mentioned in [DeMi 78, p. 130] to the insertion of the encrypted company name in software. Most such methods work either poorly or not at all when the opponent has complete knowledge of the system and arbitrary computing power. However, such an opponent cannot protect himself from the statistical methods discussed in this article, unless he can discover the true *un*fingerprinted data.

The DBMS inserts fingerprints by slightly altering or "tweaking" selected data items using a pseudo-random function of the user-id together with other identifiers. Thus each user receives a consistent view of the database, while the fingerprint insertions will vary pseudo-randomly from user to user. (See section two below.)

Many data items will not permit alterations (*e.g.*, a character string or a key attribute), while for others the permitted modification must be chosen carefully. Discussion of the difficulties of data selection and modification appears in sections two and three.

Because the fingerprints are data modifications, the opponent may try to evade identification by further modification of the data. This tactic will not work, though. Given a sufficient amount of misused data and a bound on the degree of additional modifications, the statistical methods of evaluation discussed below in section four will identify the opponent with any desired degree of certainty (*e.g.*, a 95% confidence level). In fact, given specific sizes for the fingerprint modifications and a specific bound for the opponent's further modifications, section four produces a specific number of required misused data items to guarentee a desired confidence level. Even without a bound on the opponent's modifications, the methods of section four will produce an estimate of such a bound. Section five below gives an extended hypothetical example of the application of these statistical methods.

The opponent could use especially large additional data modifications, so that identification would require much more misused data, but such additional tampering may render the data useless. It further skews the data which the DBMS has already modified within acceptable ranges.

Some users will need access to the unmodified database. For example, a court of law deciding a criminal case should not do so with fingerprinted data. Section six discussed different levels of fingerprinting, including an unfingerprinted view. Section six also presents a method that would allow

[1]This material is based in part upon work supported by the Texas Advanced Research Program.

CH2840-7/0000/0330$01.00 © 1990 IEEE

identification of coalitions of users leaking data and would allow more rapid statistical analysis after a leak.

The use of fingerprints does not directly prevent the misuse of data. This is an unavoidable condition, since we assume that the opponent has legitimate access to the data. Instead, fingerprinting acts as a deterrant to misuse, and may alert the innocent user that he has been compromised. This approach only guards against leaks of specific data items—leaked values such as averages would not be as well protected (unless the average is a derived attribute that is itself fingerprinted). To our knowledge, a system such as we describe has not been implemented or used.

This approach to fingerprinting uses a pseudo-random code with no explicit simple decoding algorithm. Such a code makes sense in this setting since one would mainly encode or fingerprint data, and would only rarely decode (*i.e.*, identify the opponent in the event of misuse). Fingerprint insertion (encoding) is simple, while decoding is aided by its use of only that small portion of the entire data set returning as misused data.

Misuse of legitimately obtained data will remain a security problem even if all other technical problems have been solved. Fingerprinting seems the only solution to this final problem. Our system will catch any opponent who misuses a sufficient amount of data. One need not assume a consistent or continued behavior by the opponent. The opponent himself would be able to monitor his increasing exposure as he leaks data, but he has only two choices: stop leaking data or risk increasingly certain detection.

2. Fingerprinted Views

A *view* in a relational DBMS is a set of stored or derived relations, expressed in a data definition language [Denn 86], [Ullm 88]. Often, a view functions as a mask, presenting only those fields relevant to a particular objective. Intuitively, a view or an *external schema* [Lars 86] is "an abstraction of part of the conceptual database" [Ullm 88, p. 9]. Thus a *fingerprinted view* is a view with small modifications to a subset of the data items.

Fingerprints must be consistent for a given user, even if a given data item appears more than once or in combination with other data items. For this reason one should fingerprint only dependent attributes of a single relation in a normalized relational database. The fingerprints must not interfere with any semantics behind an attribute, *e.g.*, if specific numerical ranges of an attribute have some semantic meaning, one should either not fingerprint the attribute at all or else not allow large enough modifications to move out of the logical range.

In constructing fingerprinted views, one faces problems similar to those in the enforcement of multilevel security [Clay 83], [Denn 85], [Grau 85]. A simple "front end" might be used for fingerprinting, placed between the DBMS and the users, like the *trusted filters* of multilevel security. This approach leaves the front end simple enough for formal security proofs, but it is then difficult to handle selections, projections, statistical queries, and other advanced features of a relational database. A better approach, taken for multilevel security in [Denn 86], places features such as security enforcement (and fingerprinting) at a low level in the conceptual database, just above the physical database. Such additions to a DBMS should still have a tractable model and formal correctness proof.

The view provided to each user must appear complete as well as consistent. Completeness may be accomplished in two ways: *globally* (fingerprinting the entire database, separately for each user) and *by request* (applying fingerprints as data values are requested). Global completeness requires an unacceptable storage overhead even for a small database. Instead, implanting fingerprints by request produces a *virtual conceptual database*, with fingerprints specific for the user, applied as the data values are needed. This eliminates the storage requirements, but additional CPU processing costs remain. See Figure 1 for a simple example.

This article concentrates on *binary* fingerprints placed on *numeric* attributes (and attributes like dates). The fingerprinted data items will have two possible values, modified and actual, one of which is chosen for a user's fingerprinted view. Our approach assumes that each fingerprinted attribute will have a delta value D (positive or negative) in the database description. Each individual user will consistently find either the actual value or the actual value plus D in his fingerprinted view. If the opponent compromises several users and obtains two versions of a fingerprinted datum, he still does not know which is the true value.

Our method of fingerprint insertion using a fixed positive or negative delta has the disadvantage that if a sum or average of values is computed, the fingerprints accumulate, since they have the same sign. As an alternative, one could use a fixed positive delta which is either added or subtracted for the fingerprint. Though the opponent may gain slight advantage in determining the actual data, averages and sums are close to the true values.

Any method that allows choosing one of two (or more) alternative data values would work in our approach. We use a delta value to insert binary fingerprints on numeric attributes for simplicity—particularly in the database description, where only the extra delta value needs to be decided upon and stored. The main criterion is that the opponent should have no way to favor one data value over another. In an extreme example, the database could explicitly store two possibilities for each attribute value. This approach might involve too much work in deciding on two possible values, and it would increase storage requirements.

The choice of which version of the data item to issue is a function of the *user-id*, the *relation name*, the *attribute name*, and the *key value*. (The fingerprint might also depend on the *group-id*, and the *user level* as discussed in section six.)

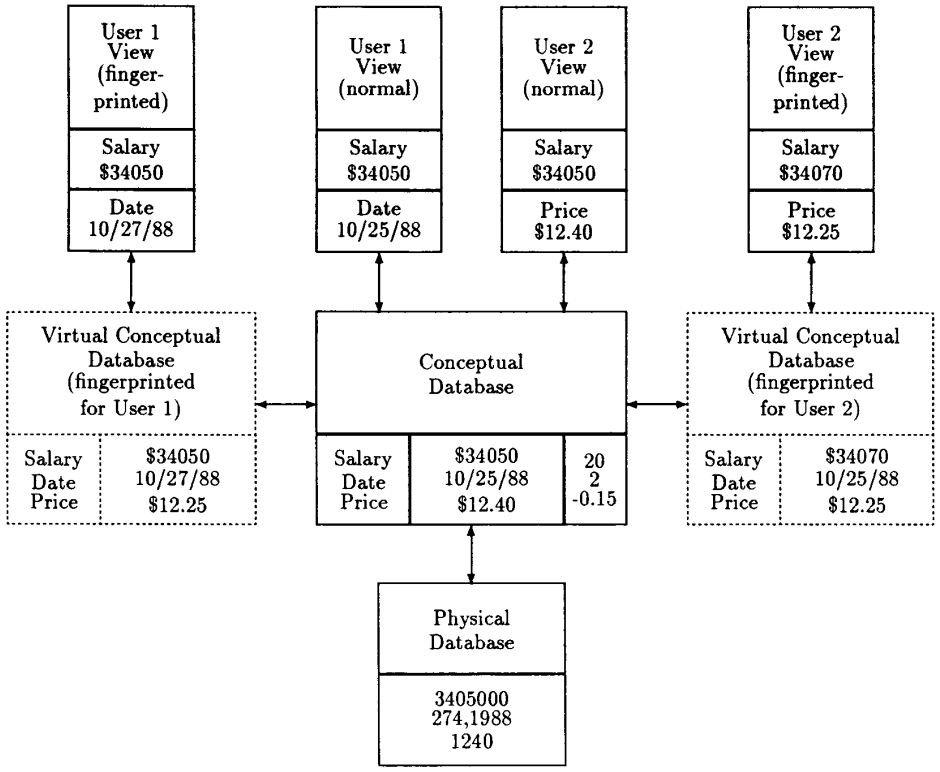

Figure 1. Fingerprinted Views with Virtual Conceptual Databases.

The database will employ a *pseudo-random hash function*, which uses the concatenation of these items as a seed to produce a binary result, choosing either the actual or the modified data item.

3. Details of Implementation

The feasibility of fingerprint implementation is not obvious. It may be difficult to find dependent attributes whose values can be modified at all. Even for simple modifiable numeric fields there are difficulties in setting a tolerance (the delta value mentioned in section two). We admit that fingerprints do distort the data—there is obviously some effect. However, small changes can often be made with minimal problems, as illustrated in this section and in section five.

As an example, consider a medical database. Medical records have often been revealed inappropriately, though access to these records is essential during an emergency. A medical record has many data items that could be fingerprinted by a medically knowledgeable DBMS administrator. For instance, the exact date for a Rubella vaccination is not important to a doctor, and could be modified in our system if one is willing to regard a date as a *numeric* attribute. The systolic portion of blood pressure varies several points even at rest. A change of a few points to a standardized test score should not alter an overall evaluation. Users should be told that fingerprints are present so that they could expect slight variations from correct values.

Consider as a second example Scholastic Aptitude Test (SAT) scores for college entrance in the United States. Some parents and students would react very negatively to the lowering of an SAT score by even a few points. They might feel that those few points could make a crucial difference. (Of course they would also object to *raising* someone else's score.) This example illustrates the tradeoff between security and accuracy. One needs to convince users to tolerate fingerprint changes for increased security.

It may be difficult to find suitable properties to alter. For example in the database of a U. S. Federal Reserve Bank, the data consists of interest rates, par amounts, id numbers, bank numbers, account balances, number of treasury tenders

purchased, and so forth. There does not appear to be any property here that can be fingerprinted. Banks are poor candidates for fingerprinting partly because their data is not secret—banks require authentication. Still, account balances and various dates could be modified if users understood that fingerprints were present.

The *date* fields mentioned above are often good candidates for modification. But here one should not necessarily use a simple delta value of a fixed number of days. One might make the date of a loan a Sunday rather than a weekday. Thus modifications to dates might take weekdays, and perhaps holidays and other important dates into account. This could be done by software with a built-in calendar, replacing a fixed delta value by an algorithm.

Other fields also require special considerations regarding delta values. An arbitrary delta value applied to a salary field might change it from \$36,000 to \$36,001.13, making the fingerprint obvious. Fields such as salaries or loan amounts would need a "round" delta value. In general the semantics behind an attribute can make problems for our methods.

The fixed delta value should be stored in the database description. The modifications needed to the DBMS software to handle fingerprinting would be straightforward: low level DBMS routines retrieving data from the physical database for the conceptual database would apply the fingerprints. For each access of a fingerprinted data item, a binary-valued hash function needs to be computed, as mentioned in section two:

```
IF hash(UserId, RelationName, AttributeName,
        KeyValue) = 1 THEN
     return(AttributeValue + AttributeDelta)
ELSE return(AttributeValue)
```

This sounds like quite a bit of computation, but such a hash function could be computed quickly with special hardware, using for example a Data Encryption Standard (DES) chip and cipher feedback mode [Denn 82, p. 147]. (Roughly the same amount of computation is involved in storing a database in encrypted or compressed form, and this is now done frequently.) The specific hash function used would need to be kept secret—for example based on a secret DES key. If one is taking an average over a fingerprinted attribute, then each separate value participating in the average would get an individual fingerprint, and the average would thus be consistent with other similar data for a given user. (Because of this consistency, our fingerprints do not give any extra security against statistical database attacks [Denn 82], except that any deduced value might be a delta away from the true value.)

4. Statistical Analysis of Misused Data

This section uses the binary fingerprints discussed earlier and a specific statistical algorithm for processing any returned data. There are other approaches, but this one works well. The algorithm has the advantage that any bias of the opponent's modifications will cancel out during the calculations. Readers not interested in this statistical analysis should skim through the discussion to the algorithm and examples at the end of the section.

In order to use the proposed statistical analysis, assumptions must be made regarding the opponent's behavior. These assumptions will be described, and are summarized below in equation (2). The analysis of the data then takes the form of a widely used statistical procedure, the two-sample test for equality of means. (Not all details of the theory are presented here—the interested reader can write to the second author for such details.)

In all statistical hypothesis tests there are two types of error possible. In the present context, the first type of error would conclude that a particular user is the opponent, when he is not. The second would conclude that a user is not the opponent, when he actually is. The probabilities (denoted below by α and β, respectively) of committing the two types of errors can be made as close to 0 as desired, although they can never be completely eliminated.

Suppose we have N real *data values* $V_1, V_2, \ldots, V_N$ and m *users*. Each value V_j must have an associated *delta value* $D_j > 0$ with the property that any number in the closed interval $[V_j - D_j, V_j + D_j]$ is acceptable for use by all users.

For 50 percent of the data values, users are provided with either V_j or $V_j + D_j$, and in the remaining cases, users get either V_j or $V_j - D_j$. All choices are done in pseudo-random fashion as described earlier. With this strategy, any coalition of users has at most two versions of a data value to work with—one correct and one incorrect, and they do not know which is which. For the analysis here, one needs two acceptable values for each datum, and it does not matter whether one is "correct" and the other is"incorrect." The version of the j^{th} datum sent to User i is denoted V_{ij}.

Now suppose the data has been leaked to the press by the opponent without authorization, and that values $V'_1, V'_2, \ldots, V'_n$ appear in a staff writer's column. (Usually n will be much smaller than N above). For each i, $1 \le i \le m$, test the hypothesis that User i is the source of the leaked values. For this purpose examine the numbers

$$y_{ij} = \frac{V'_j - V_{ij}}{D_j}, \quad 1 \le j \le n, \quad \text{fixed } i. \tag{1}$$

The y_{ij} are the standardized differences between the returned values and the values given to User i.

For fixed i, consider means over two disjoint subsets of the y_{ij}. Let S_{i1} be the set of indices j for which V_{ij} is the higher of the two versions of V_j sent to different users, and S_{i2} the set of indices for which V_{ij} is the lower of the two versions. Let n_{i1} and n_{i2} be the cardinalities of sets S_{i1} and S_{i2}, respectively. Let $\bar{y}_{i1}$ and $\bar{y}_{i2}$ be the averages of the y_{ij}

over S_{i1} and S_{i2}, and let $d_i = \bar{y}_{i2} - \bar{y}_{i1}$.

Suppose that $V_1', V_2', \ldots, V_N'$ are independent random variables with V_j' having mean $V_{ij} + \delta_j$ and variance τ_j^2. Also assume that the amount of "tweaking" done by the opponent is proportional to the amount done by the DBMS administrator. Letting μ and σ^2 represent the constants of proportionality, these assumptions may be stated as

$$\frac{\delta_j}{D_j} = \mu \quad \text{and} \quad \frac{\tau_j^2}{D_j^2} = \sigma^2. \tag{2}$$

The usual two-sample test for equality of means can now be carried out. The following two hypotheses will be tested:

H_0, the *null* hypothesis: User i is the opponent.
H_1, the *alternative* hypothesis: User i is *not* the opponent.

The following quantity, called the *test statistic*, is then calculated:

$$z_i^* = \frac{d_i}{\sqrt{\frac{s_{i1}^2}{n_{i1}} + \frac{s_{i2}^2}{n_{i2}}}}, \tag{3}$$

where

$$s_{ik}^2 = \frac{1}{n_{ik} - 1} \sum_{j \in S_{ik}} (y_{ij} - \bar{y}_{ik})^2, \quad \text{for} \quad k = 1, 2. \tag{4}$$

According to the Central Limit Theorem for non-identically distributed independent random variables [Brei 68], the quantity z_i^* will have an approximate standard normal distribution, provided that n_{i1} and n_{i2} are large. The null hypothesis is rejected if $|z_i^*|$ exceeds z_α, the upper 100α percentage point of the standard normal distribution. These z_α values are readily available in statistical references such as [Koop 81] or [Owen 62]. For the reader's reference, we include a short list of (α, z_α) pairs: $(0.1, 1.282)$, $(0.05, 1.645)$, $(0.01, 2.326)$, $(0.005, 2.576)$, $(0.001, 3.090)$, $(0.0005, 3.290)$, $(0.0001, 3.720)$, $(0.00005, 3.890)$, $(0.000005, 4.420)$.

The probability of rejecting H_0 when H_0 is actually true is α, which can be chosen as small as desired. The probability of accepting H_0 when H_0 is actually false depends upon the true identity of the opponent. Let β represent the probability of accepting User i as the opponent when User k ($k \neq i$) is the true opponent. It can be shown that

$$\beta \doteq \Phi\left[z_\alpha - \frac{\sqrt{n}}{2\sigma}\right], \tag{5}$$

where $\Phi(\cdot)$ is the cumulative standard normal distribution function. Now the sample size required to achieve a specific α and β is

$$n = 4\sigma^2(z_\alpha + z_\beta)^2. \tag{6}$$

The z_α and z_β are read from a table of the standard normal distribution, as mentioned above. The variance, σ^2, may be estimated using prior information, or after the data have been collected, it may be estimated by a weighted average of the sample variances (4) as follows:

$$\sigma^2 = \frac{(n_{i1} - 1)s_{i1}^2 + (n_{i2} - 1)s_{i2}^2}{n_{i1} + n_{i2} - 2}. \tag{7}$$

Figure 2 below summarizes the relations between α, β and n, using normal distributions centered at 0 for the null hypothesis and centered at $\frac{\sqrt{n}}{2\sigma}$ for the alternative hypothesis. Notice that as the sample size n increases, the curve centered at $\frac{\sqrt{n}}{2\sigma}$ moves to the right, decreasing β rapidly.

Figure 3 on the next page summarizes the work so far as an explicit algorithm.

EXAMPLE 1. Suppose that α and β are required to be 0.01 each. Assume that $\sigma^2 \doteq 1$, that is, the opponent "tweaks" the values by about the same amount as the DBMS administrator. Then $n = 4(z_{0.01} + z_{0.01})^2 = 4(2.326+2.326)^2 = 86.56$. So roughly 90 values would have to be observed. The test statistic z_i^*, as described by equations (3) and (4) above, would then be calculated. If z_i^* exceeded $z_{0.01} = 2.326$, User i would be declared innocent. This test would be carried out for each i, $1 \leq i \leq m$. Note that it is possible for several users to be accepted as the opponent. This effect can be minimized by choosing a small β and increasing the sample size as required. If α and β were set equal to 0.0001, then $n = 4(z_{0.0001} + z_{0.0001})^2 = 4(3.720 + 3.720)^2 = 222$. If the opponent were to "tweak" values by roughly twice the amount of the administrator, then one would have $\sigma^2 \doteq 4$, and the values of n above would need to be nearly 350 and nearly 900 for the two examples above.

EXAMPLE 2. In a previous article [Wagn 83] the first author presented a simulation with $n = 200$ returned values, and with 40 separate users. Here the ratio of the opponent's changes to the fingerprints was $\sigma = 1.233$. If processing one user, the α and β may each be taken to be 0.002 with this number of returned items. Thus for any single user, we have probabilities of 99.8% of correctly identifying that user as innocent or as the opponent. After processing all 40 users, a conservative estimate of the probability of identifying the opponent is $1 - 40\beta$, or about 92%.

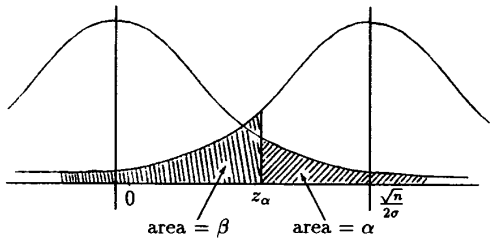

Figure 2. Illustration of the Hypothesis Test. ("Bactrian" curves.)

ALGORITHM. *Test the hypothesis that User i is the opponent.*

1. Choose an acceptable probability α: that the algorithm will fail to identify User i as the opponent, when he is and an acceptable probability β: that the algorithm will identity User i as the opponent, when he is not. Use equation (6) to calculate the necessary sample size, n.
2. Use the returned data $\{V_j'\}$, the original data sent out $\{V_{ij}\}$ and the delta values $\{D_j\}$, each for $1 \leq j \leq n$, to calculate
   - y_{ij}: standardized differences, using equation (1),
   - $\bar{y}_{i1}$ and $\bar{y}_{i2}$: averages of y_{ij} over S_{i1} and S_{i2}.
   - d_i: $\bar{y}_{i1} - \bar{y}_{i2}$.
   - s_{i1}^2 and s_{i2}^2: the sample variance of y_{ij} over S_{i1} and S_{i2}, using equation (4).
   - z_i^*: the test statistic, using equation (3).
3. If $z_i^* > z_\alpha$, then the algorithm concludes that User i is not the opponent. Probability of error is $\leq \alpha$.
4. If $z_i^* \leq z_\alpha$, then the algorithm concludes that User i is the opponent. Probability of error is $\leq \beta$.
5. At this point, σ^2 may be estimated by equation (7). If this estimate exceeds the original estimate of σ^2, the necessary sample size should be recalculated using (6). (This means either waiting for more data values to arrive or accepting a higher value of β than was chosen in step 1.)

Figure 3. Fingerprint Identification.

From the above discussion it should be clear that the opponent's best strategies are to leak very little data and to change it as much as possible. As the amount of misused data increases, chances of detection increase in proportion to the percentage of fingerprinted data in the data set. Security personnel may select the level of confidence they desire in analysis and proceed. In any event, the opponent's effectiveness in evasion coincides to his ineffectiveness as an opponent.

5. Extended Example

For a more realistic and lengthy example, consider the counseling department of a large corporation, which keeps a database of employee's scores on the Minnesota Multiphasic Personality Inventory (MMPI) [Lach 74]. Suppose that the scores are meant to be confidential, but that someone is obtaining the scores in some unauthorized way and using this information in a harmful manner. For example, a manager bases his decision not to promote a certain employee on the fact that the employee had an undesirable personality profile. After this is suspected of occurring several times, the DBMS administrator decides to try to do something about it.

The administrator consults with a professional psychologist to decide on a delta value for the fingerprints on scores. On each part of the test, the normal range is a region 15 points wide, with the adjacent higher and lower regions being 10 points wide. Evaluations are based in part on the overall shape of the plot of the scores on the thirteen parts of the test; specifically, combinations of "peaks" in the graph at certain locations are important. The administrator and consultant decide that score values altered by ± 2 points should not result in any significant variation in the overall evaluation.

There are 50 different professionals who have legitimate access to the data, and access is controlled tightly, so that it is felt that either the opponent is himself a user, or has compromised one of the 50 legitimate users. A secretary agrees to copy down as many leaked test scores as possible, without the manager's knowledge or permission. It appears at first glance that the opponent has simply been rounding all scores to the nearest 5 points. The ratio of the variance of the opponent's tweaking to the variance in the fingerprints is 2 to 4, so σ^2 is 0.5. (The variance is the average squared deviation from the true value, and rounding to the nearest five points results in deviations of 0, 1, 2, -2, and -1.) Values of α and β equal to 0.0001 are chosen so that the sample size n needed to achieve this, given by equation (6) is $n = 4\sigma^2(z_\alpha + z_\beta)^2 = 4(0.5)(3.72 + 3.72)^2 = 111$. The strategy is to wait for 111 returned score values, or approximately 9 complete MMPI scores, and then do the statistical processing. At that point equation (7) will be used to re-estimate σ^2.

However, after providing 6 sets of scores, or 78 values, to the DBMS administrator, the secretary quits due to illness. Now the administrator decides to use the available values for the statistical analysis, even though there are not as many as originally planned. With this value of n, values α and $\beta = 0.001$ can still be achieved.

For each individual test, we have a 0.001 chance of falsely accusing a user or of missing the true opponent. After checking each of the 50 users, we have a probability of at least $1 - 50(0.001) = 95\%$ of identifying the correct user, and only that user, as the opponent. Then the hope is that one can use other means to prove that this user is indeed the source of the leak.

6. Refinements

In the case of a large number of users, identifying the opponent could be a lengthy computation. The method in section four has time complexity $O(nm)$, where n is the number of returned fingerprinted data items and m is the number of users. One could introduce a *group-id* in addition to the user-id. (This group-id might simply identify the organization of

the user.) Based on a binary pseudo-random hashed value of the relation name, attribute name, key value, and group-id, one decides whether or not to insert individual fingerprints or group finger prints. Thus half of the data items receive a group fingerprint, identical for every user in the group. Identification of the opponent proceeds in two stages, first identifying the group and second identifying the individual within that group. The time complexity of the two-stage process is $O(n\sqrt{m})$ with the same implied constant. However, one now needs approximately twice as many returned fingerprinted items for the same degree of confidence in the identification.

An additional advantage of group-ids is that a coalition of users from the same group would not be able to notice approximately half of the fingerprints. In this way, one would be able to identify the origin group of a security breach, and know which organization has been compromised.

As mentioned in section three, it will often be necessary for some agencies to receive a view with no fingerprints. One could introduce a *user level number*. The database description would include extra information indicating which items to fingerprint and giving the specific delta value to use for each level number. Level 0 would imply "no fingerprints." Progressive levels would have sloppier fingerprints. In the simplest case, one could have two levels: one with, and one without fingerprints.

7. Conclusions

A fingerprinted database will not directly prevent misuse of data, but will allow detection of the source of misused data and will hence act as a deterrent. In the event of a data leak from a legitimate user, a statistical analysis will identify the user, while routine fingerprint insertion can be carried out during data accesses. Though fingerprinting introduces difficulties in protecting some types of data, in general it functions to reduce data leaks since it behooves the opponent to reduce the amount of data that he leaks. Continued leaks produce a cumulative trail which will eventually reveal the source of insecurity.

Acknowledgement

Elaine DeBrine, a clinical psychologist, gave technical assistance for the example in section five.

References

[Brei 68] L. Breiman, *Probability*, Addision-Wesley, 1968.

[Clan 87] T. Clancy, *Patriot Games*, Berkley Publishing Corporation, 1987 (paperback edition, 1988).

[Clay 83] B. G. Claybrook, "Using views in a multilevel secure database management system," *Proceedings of the 1983 Symposium on Security and Privacy*, IEEE Computer Society, 1983, pp. 4–17.

[DeMi 78] R. A. DeMillo, et al., *Foundations of Secure Computation*, Academic Press, 1978.

[Denn 82] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley Pub. Co., 1982.

[Denn 85] D. E. Denning, et al., "Commutative filters for reducing inference threats in multilevel database systems," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, 1985, pp. 134–146.

[Denn 86] D. E. Denning, et al., "Views for Multilevel Database Security," *Proceedings of the 1986 Symposium on Security and Privacy*, IEEE Computer Society, 1986, pp. 156–172.

[Grau 85] R. D. Graubart, and K. J. Duffy, "Design overview for retrofitting integrity-lock architecture onto a commercial DBMS," *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE Computer Society, 1985, pp. 147–159.

[Koop 81] L. H. Koopmans, *An Introduction to Contemporary Statistics*, Duxbury Press, 1981.

[Lach 74] D. Lachar, *The MMPI: Clinical Assessment and Automated Interpretation*, Western Psychological Services, 1974.

[Lars 86] J. A. Larson, *Database Management*, IEEE Computer Society, 1986.

[Owen 62] D. B. Owen, *Handbook of Statistical Tables*, Addison-Wesley, 1962.

[Ullm 88] J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Vol. I, Computer Science Press, 1988.

[Wagn 83] N. R. Wagner, "Fingerprinting," *Proceedings of the 1983 Symposium on Security and Privacy*, IEEE Computer Society, 1983, pp. 18–22.