Security of Relational Databases in Business Outsourcing

Ersin Uzun, Bryan Stephenson

HP Laboratories
HPL-2008-168

Keyword(s):
database watermarking, fingerprinting encryption

Abstract:

For most corporations the volume of sensitive data used by outsourcing providers continues to increase. As the number of different entities having access to a database increases, it gets harder to prevent and trace-back data leakage. We address the problems of proving ownership and unauthorized data distribution (leakage) for relational databases. We propose three techniques that altogether may be used to detect, deter and trace-back data leaks from relational databases. We use business process outsourcing scenarios as the descriptive use case, but our techniques are equally applicable in other use cases when a relational database is shared among many parties and its confidentiality and authenticity needs to be protected. Previous work has shown how to watermark and fingerprint numerical relational data to prove ownership and track unauthorized redistributions respectively. This work represents the first attempt to find more general solutions that can practically accommodate relational data with non-numerical or error sensitive attributes that are common in corporate databases.

External Posting Date: October 21, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: October 21, 2008 [Fulltext]

Security of Relational Databases in Business Outsourcing

Ersin Uzun[1,2] and Bryan Stephenson[2]

[1] University of California, Irvine CA 92697, USA,
[2] HP LABS, Palo Alto CA 94306, USA

euzun@ics.uci.edu, bryan.stephenson@hp.com

Abstract. For most corporations the volume of sensitive data used by outsourcing providers continues to increase. As the number of different entities having access to a database increases, it gets harder to prevent and trace-back data leakage. We address the problems of proving ownership and unauthorized data distribution (leakage) for relational databases. We propose three techniques that altogether may be used to detect, deter and trace-back data leaks from relational databases. We use business process outsourcing scenarios as the descriptive use case, but our techniques are equally applicable in other use cases when a relational database is shared among many parties and its confidentiality and authenticity needs to be protected. Previous work has shown how to watermark and fingerprint numerical relational data to prove ownership and track unauthorized redistributions respectively. This work represents the first attempt to find more general solutions that can practically accommodate relational data with non-numerical or error sensitive attributes that are common in corporate databases.

1 Introduction

Demanding market conditions encourage many companies to outsource certain business processes (e.g. marketing, human resources) and associated activities to a third party. This model is referred as Business Process Outsourcing (BPO) and it allows companies to focus on their core competency by subcontracting other activities to specialists, resulting in reduced operational costs and increased productivity. Security and business assurance are essential for BPO. In most cases, the service providers need access to a company's intellectual property and other confidential information to carry out their services. For example a human resources BPO vendor may need access to employee databases with sensitive information (e.g. social security numbers), a patenting law firm to some research results, a marketing service vendor to the contact information for customers or a payment service provider may need access to the credit card numbers or bank account numbers of customers.

The main security problem in BPO is that the service provider may not be fully trusted or may not be securely administered. Business agreements for BPO

try to regulate how the data will be handled by service providers, but it is almost impossible to truly enforce or verify such policies across different administrative domains. Due to their digital nature, relational databases are easy to duplicate and in many cases a service provider may have financial incentives to redistribute commercially valuable data or may simply fail to handle it properly. Hence, we need powerful techniques that can detect and deter such dishonest or careless behavior.

Watermarking and fingerprinting in the context of detecting and deterring unauthorized distribution of copyrighted multimedia data has been studied extensively, e.g., [7, 8, 13, 14]. However, not much work has been done on watermarking and fingerprinting relational databases. Although some recent results [3, 2, 1, 23, 16, 12, 11, 15] have shown how to watermark or fingerprint relational databases with numeric attributes, their assumptions are too strong to be useful in many real-life scenarios. They assume that there exists at least one *numerical attribute* in a database satisfying the following two properties: (1) the value of the database would be significantly lower without it (2) the values for that attribute are tolerant to introduced errors. In other words, none of the existing methods is applicable when

- there are no numerical attributes in the database, or,
- a part of the database still carries meaningful and valuable information when numerical attributes are separated from the rest of the database, or,
- modifying the numerical values to embed a watermark is not acceptable. This actually is a stronger limitation than one would first think of, since such changes may be detectable or may disrupt the business process in many cases [3].

In this paper, we introduce two new watermarking techniques that relax the assumptions set by the previous work and yield to a wider application domain. For both techniques, we also show how they can be extended to a collusion-secure fingerprinting scheme for small distribution groups.

Our first technique employs permutation of certain attribute values among deterministically chosen tuples. Unlike previous methods, this technique can accommodate databases without any numerical attributes as well as those with attributes in which the error introduced by the mark can be detected. The

[3] Previous methods modify a portion of tuples and introduce errors on the least significant bits of the numerical values to embed a mark. This implies 3 assumptions about the numerical data: (1) it is OK to modify the numerical values on a portion of tuples (Some numerical corporate data like account numbers are not tolerant to even the smallest errors), (2) extensive modifications on the least significant bits throughout the database (e.g., rounding all the numbers) would make the data worthless, (3) tuples holding a mark, i.e., having random looking errors introduced by the marking algorithm, cannot be detected by an outsider (Often numerical corporate data doesn't satisfy this assumption (credit card numbers have a check digit for easy error detection, and changing a model number for a Boeing plane from 747 to 746 is far from being undetectable).

shortcoming of this method is that it is only applicable to databases that are tolerant to errors introduced on a small fraction of its tuples.

In the second technique, we weaken our assumption of error tolerance from errors on the real data originally residing in the database to errors introduced by inserted virtual tuples. Note that adding a small number of extra tuples doesn't cause any error or loss on the original tuples. This is an important distinction since many business processes cannot tolerate errors on the real useful data but they can tolerate some extra data that is not real. For example if a database of customer addresses is used to mail marketing literature, it could include some artificial addresses for marking purposes and still all real customers would receive the literature in the mail. In this case, some of the additional records can also be used to verify that the literature was indeed sent by the service provider. We also discuss how these extra tuples may be used as honey pots to detect leaks in a timely manner and how the negative impact of having extra tuples can be eliminated in certain BPO scenarios.

Watermarking and fingerprinting techniques may encourage service providers to handle data more carefully as they can be linked to data leaks, but they are not preventative measures and data may still end up in the wrong hands under many scenarios where the service provider has no bad intentions. For instance, a new vulnerability in an operating system may allow a hacker to gain access to the data, or an angry employee may post it online. To improve security, we advocate incorporating proactive and reactive measures if possible, and in this respect we developed a third complementary method that can be used to prevent data leaks from happening.

Under certain, though very limited, scenarios, we observed that the service provider may not actually need to see specific attributes of a database for it to be usable. For example an online payment service vendor needs credit card numbers just to take them to credit card companies to clear a charge of a certain amount. Here the service provider has no reason to see the actual card number if the credit card company can still charge the correct account and the service provider gets its money. Hence, the third technique we developed is an encryption based protocol. It can be used when certain sensitive database attributes (e.g, SSN, credit card numbers), which are governed by very few entities (e.g., government, credit bureaus, credit card companies), are handed to a service provider. In such cases, the data may be kept hidden from the service provider by encrypting it from the data owner to the governing entity using public key cryptography and asymmetric encryption. Whenever applicable, this method can be incorporated with watermarking and fingerprinting techniques to increase the security of highly sensitive information. It is also important to note that the encryption could be the only choice on protecting highly error sensitive databases that cannot be watermarked or fingerprinted.

In the rest of the paper, we review the related work in section 2. In section 3 we describe our three methods to watermark, fingerprint, or encrypt data. We discuss practicality of the methods in section 4 and conclude in section 5

2 Related Work

There has been quite extensive research on fingerprinting and watermarking multimedia data, e.g., [7, 8, 13, 14]. Unfortunately these techniques do not practically extend to relational data. The main obstacle in applying multimedia watermarking techniques to relational databases is the significant difference between the data properties. Multimedia data has high redundancy and its segments are correlated, while databases are redundancy free sets where each of its tuples can be manipulated independently. Multimedia data is meaningful when its segments are in correct order but each database tuple constitutes a meaningful piece by itself. Unlike multimedia data, databases are also subject to continuous change due to tuple addition, deletion, and other modifications throughout their life cycle.

Researchers have proposed solutions for watermarking and fingerprinting of computer programs [6, 22, 5, 19, 9] and text [4, 20]. However, those techniques are based on either hiding a mark in the visual representation or transforming data into a semantically equivalent alternate representation, hence they are not applicable to databases.

Watermarking and fingerprinting of relational databases has been pretty active in the last couple of years yielding to many proposals [3, 2, 1, 23, 16, 12, 11, 15, 17, 18]. Although these techniques are shown to be effective on databases with fairly error tolerant numerical attributes, they don't extend to many databases that are common in the business world. For example, human resources or customer databases may often have numerical values (e.g., SSN, credit card or phone numbers) but those numbers are not usually error tolerant. Moreover, if non-numerical or non-error-tolerant attributes are independently valuable, then any markings introduced on the other attributes would not be effective as the attacker can easily strip the markable parts off the database and sell the remaining –still valuable– part.

The amount of non-numerical data stored in relational databases is considerable and there has been very little work on watermarking these databases. Although some of the above techniques proposed for numerical data, e.g., [10], may be trivially extended to perturb letters instead of digits, it would not be secure as such modifications are usually reversible by automated tools like spelling checkers.

Contrary to the previous work, which has strict assumptions about the data properties, this paper introduces more general solutions with weaker data assumptions. The techniques described here are applicable over wide range of data types commonly found in corporate databases and can provide security assurance in various business process outsourcing scenarios, especially when the previous solutions are not applicable.

3 Methods to Secure Relational Databases in BPO environments

In this section, we introduce two new watermarking schemes for relational data, probabilistically analyze their security and performance, and describe how each scheme can be extended to embed unique collusion-secure fingerprinting codes into different copies for each recipient. Complementary to the first two techniques, we also describe a third scheme in section 3.3 that is applicable in certain BPO scenarios to provide preventative security measures against data leakage.

3.1 Permutation Based Watermarking

Although many watermarking schemes embed a secret mark by introducing certain errors and changes on the attribute values of a database, this approach is only effective as long as those changes are not detectable by third parties. If the changes can be detected by third parties, the embedded mark can be easily removed.

Thanks to public services over the internet and software tools available today, a lot of information can be automatically checked for errors once it is acquired. If automated error checking can be done for many or all valuable attributes in the database, any technique introducing errors inside attribute values would fail to be secure. Common examples of such easily verifiable attributes are mailing addresses (U.S. addresses can be checked for errors and be standardized using automated tools [21]), names (most names can be automatically checked using name dictionaries or the number of page hits in Google), natural language components (natural language can be automatically checked for spelling errors) and many other string based values that can be verified through searches on internet search engines. Due to this issue, the application domain of the previous work is limited to numerical values that cannot be automatically checked for errors, like experimental measurement data.

In some other scenarios, even if it will be undetectable, modifying numerical data at the level of individual attributes may not be desirable if the statistics (e.g., min, max, sum, mean, standard deviation, distribution model) for a corresponding column are important to be kept precise for future analysis. Previous solutions are not capable of keeping all the statistics over a column of a database intact.

To address the problem of watermarking databases when changes within attributes are undesirable, we developed a simple yet effective watermarking method based on attribute permutation. The idea behind our method is to permute few attributes on the same column of the database among different tuples while keeping the data at each single attribute unchanged. If the data is numerical, such a change doesn't introduce any change in column statistics. And, if each attribute value can be automatically checked for errors, this approach does not introduce any weakness as the individual attribute values are kept untouched but just few of them are mislocated among tuples.

Although the idea looks simple, choosing the attributes to be permuted and how to permute them has challenges. The four points that need particular attention are

- Adjustable security: The percentage of the attributes that will be permuted should be adjustable for different security needs while permutations should result in an acceptable ratio of erroneous tuples.
- Blind detection: The scheme should allow detection of a mark on a database without keeping the original database or a log of committed changes. Blind detection increases efficiency and makes the system more secure against database updates and attacks like subsetting[3].
- Minimize errors: During permutation, the error introduced on individual tuples should be minimized. In other words, if an attribute in a tuple should be replaced, it should be replaced with the one that is closer to its original value according to a predefined distance function (e.g., euclidian distance). For example; if a set of temperature values {....,70,...,15,..69,..} are to be permuted, replacing 70 with 69 should be preferred to replacing it with 15.
- Correlated attributes: Some attributes are correlated, and so a collection of correlated attributes may need to be permuted to keep the watermarking scheme irreversible. For example, if an email address field will be permuted, but a name field is also included in the database, it is advisable to also permute the name field since many email addresses are of the form firstname.lastname@domain.com, and obvious discrepancies between these two fields can make the marks detectable.

Watermarking Algorithm Given a data set D of size S, let $t.k$ be the primary key and $t.a$ be an attribute other than the primary key for a tuple t in D. For a given n bit marking key $K \in \{0,1\}^n$, cryptographic hash function $H()$, marking intensity parameter P $(0 < P < 1)$, and distance function $F(x, y)$, our watermarking algorithm is given in algorithm 1.

As you see in algorithm 1, marking is done by first finding attributes that output 0 in modulo p, $p = \lfloor 1/P \rfloor$, when hashed with the marking key. These attributes are swapped with the same attribute in a different row in which the primary key also outputs 0 in modulo p when hashed with the same marking key. As we have a set of such attributes available, we choose the one that is closest to the old attribute it is replacing as determined by a given distance function. Note that algorithm 1 takes a trivial greedy approach that would not result in the optimal minimum for the sum of errors introduced due to the distance between the old and the new values of attributes after permutation. This is done to keep the algorithm compact and simple in the paper, but we highly recommend to employ a linear program (LP) solver that minimizes the sum of introduced errors in the real implementation.

After the marking algorithm is executed, approximately $M = S * P$ of the tuples in the resulting marked database have both their primary key and the marked column attribute output 0 in modulo p ($p = \lfloor 1/P \rfloor$) when hashed together with the marking key. Now, detection is fairly easy and can be done

Input: Data Set D, Marking Intensity P, Marking Key K, Distance Function F
Output: Watermarked Data Set D_W
$MarkSet, X, Y \leftarrow \{\}$
$D_W \leftarrow D$
$p \leftarrow \lfloor \frac{1}{P} \rfloor$
foreach *tuple* t *in* D_W **do**
  if $H(t.a, K) = 0(mod\ p)$ **then**
    insert t into $MarkSet$
  end
end
if $MarkSet = \{\}$ **then**
  return $MarkFailed$
end
foreach *tuple* X *in* D_W **do**
  if $MarkSet = \{\}$ **then**
    return D_W
  end
  if $H(X.k, K) = 0(mod\ p)$ *and* $H(X.a, K) \neq 0(mod\ p)$ **then**
    $l \leftarrow \infty$
    foreach *tuple* j *in* $MarkSet$ **do**
      if $F(X.a, j.a) < l$ **then**
        $l \leftarrow F(X.a, j.a)$
        $Y \leftarrow j$
      end
    end
    swap($X.a$,$Y.a$)
  end
end
return D_W

Algorithm 1: Watermarking algorithm

blindly. The detection algorithm (see Algorithm 2) simply goes through the tuples in the database and checks if the ratio of tuples that satisfy the marking condition is substantially more than the expected ratio of such tuples on a non-marked database. The expected ratio of such tuples on a non-marked database is P^2 since the two selected attributes in the same row must hash to 0 in modulo p, and the probability of this happening on a random attribute is P.

Input: Data Set D, Marking Intensity P, Marking Key K
Output: {0,1}
$NoOfMarks \leftarrow 0$
$p \leftarrow \frac{1}{P}$
foreach *tuple* t *in* D **do**
  if $H(t.k, K) = 0(mod\ p)$ *and* $H(t.a, K) = 0(mod\ p)$ **then**
    $NoOfMarks \leftarrow NoOfMarks + 1$
  end
end
if $NoOfMarks/S > P^2 + \varepsilon$ **then**
  return 1
end
else
  return 0
end

Algorithm 2: Detection algorithm

Analysis In this watermarking scheme, the value of P affects both the invasiveness and the provided security of the marking algorithm. As P decreases, the number of modified tuples and the provided security decreases, so the procedure becomes less invasive due to a smaller number of committed permutations. On the other hand, increasing P improves security but will result in more modifications (introduced errors) in the database. Therefore, choosing a P value according to the security needs and acceptable error margin is essential.

In the detection algorithm, a well chosen value for ε is crucial to keep the error rates at a balance. An ε value that is too small would increase false positives and a value that is too large would cause too many false negatives.

In the following bullets, we show how to choose ε and P values according to the desired error rates and analyze their effect on security and invasiveness.

– During watermark detection on a given database D of size S, to limit the probability of a false positive to f_p, the ε should be large enough to provide enough statistical significance that the observed ratio of marked tuples is not coincidental. In mathematical terms:

$$Prob\{Observed\,ratio\,of\,marked\,tuples\,in\,a\,non{-}marked\,database > P^2{+}\varepsilon\} \leq f_p$$

If the null and the alternative hypothesis are:
H_0: Observed marking ratio can be coincidental (D is not marked)

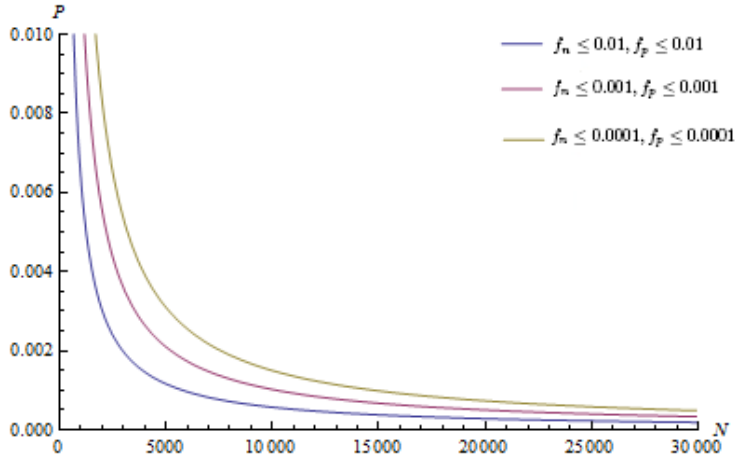

Fig. 1. Change of marking intensity parameter (P) for different minimum detection subset size (N) from 100,000 total records. Three lines corresponds to three different values for maximum allowed false negative (f_n) and false positive (f_p) rates. From top to bottom, the maximum values for f_n and f_p are set to 0.0001, 0.001, 0.01

H_1: Observed marking ratio is too high to be coincidental (D is marked)
then by setting ε large enough, we assure that H_0 will be rejected only when there is significant statistical evidence against it. If the detected number of marked tuples in a database of size S is N, the standard error rate for the proportion of marked tuples can be calculated as $\sqrt{N/S(1-N/S)/S}$. So, ε can be calculated as a function of N and S for a chosen f_p. For example, if the size of the database given into the detection algorithm is $S = 10000$, and $N = 10$ marked tuples are detected, the ε value for $f_p \leq 0.001$ would be:

$$f_p \leq 0.001 \Longrightarrow \frac{N/S - P^2}{\sqrt{N/S(1-N/S)/S}} \geq t_0 = 2.97$$

$$\Longrightarrow \frac{\varepsilon}{\sqrt{999 * 10^{-10}}} \geq 2.97 \Longrightarrow \varepsilon \geq 0.00094$$

– Similarly, choosing a large enough P value while marking a database D is important to limit the probability of having false negatives (fail to detect an existing mark). However, the number of introduced errors also increases as P increases. Hence, P should be as small as it can be afforded while still providing an acceptable false negative probability. If a random subset of size N tuples from database D is input to the detection algorithm, then limiting the probability of a false negative to f_n means:

$$Prob\{Observed\ marked\ tuple\ ratio\ in\ a\ marked\ database < P^2 + \varepsilon\} \leq f_n$$

The Distribution of the proportion of the marked tuples can be estimated by a normal distribution with mean $\mu = P$ and standard deviation $\sigma =$

$\sqrt{P(1-P)/S}$. Here P can be calculated for a desired f_n as shown in the following example. Assume a large database D is to be marked and a false negative probability of at most f_n is wanted when the detection algorithm is run on the subsets of size N or larger of D. For $f_n \leq 0.001$ and $N = 10000$, P can be calculated as:

$$P - (P^2 + 2.97\sqrt{P^2(1-P^2)/N}) \geq 2.97(\sqrt{P(1-P)/N})$$

$$\Longrightarrow P \geq 0.00094$$

Which means; if the original size of D was 100,000, marking only 94 tuples would be enough to provide the above given probabilistic guarantees. Since each marked record causes a second record to also be modified, having 94 tuples marked roughly means to modify 188 tuples (see below for more precise calculation). Figure 1 shows how P value changes with different N values when upper bounds for acceptable false negative (f_n) and false positive (f_p) error rates are set to be 0.01, 0.001, 0.0001.

– While marking a database, some attributes are permuted between different tuples to create tuples that satisfy the marking condition. However, a small number of tuples in the original database may satisfy the condition without any changes necessary. Hence, the expected ratio of modified tuples for a database is slightly lower than $2P$ and can be calculated as:

$$2 * (P - P^2) = 2P(1-P)$$

Figure 2 shows how the expected ratio of modified tuples change with P.

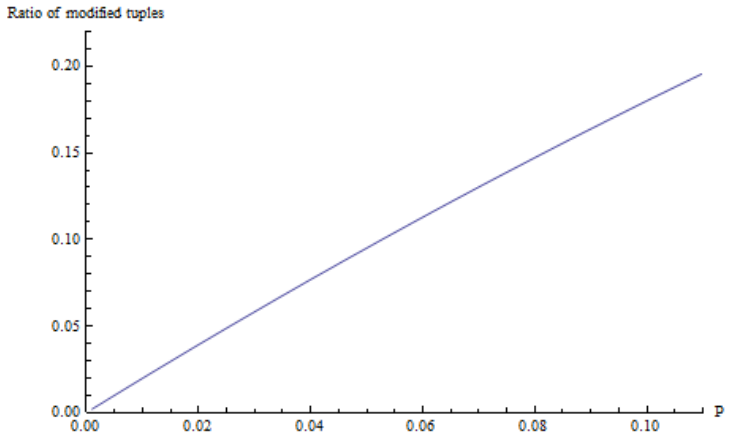

Fig. 2. The expected ratio of modified tuples for different P values

Extension to fingerprinting Although watermarking is useful to prove ownership, it is not enough to trace back the source of a data leak if more than 1 party has been given access to a particular data set. In this case, we need unique fingerprints to be embedded into each service provider's copy. If we assume K different service providers, then the length L of the binary codewords that can serve as a fingerprint should be at least $L > lg_2 K$.

Considering that most service providers are legitimate businesses, collusion secure fingerprint codes are overkill for many business outsourcing scenarios. Hence, we predict a counting based binary fingerprinting code (e.g., for 4 providers the codes will be {00,01,10,11} and for Q providers, the length L of simple binary codewords is $L = \lceil lg_2 Q \rceil$) that is not collusion secure would be secure enough for many practical BPO scenarios. However, it is worth noting that the fingerprinting algorithm introduced here works with any binary fingerprinting code (collusion resistant or not) as long as the bit-length of the codewords to be embedded is considerably smaller than SP. Nevertheless, the length of collusion resistant fingerprinting codes would increase dramatically as the number of service providers would increase, and we only see the possibility of using collusion-resistant fingerprinting codes when the number of different service providers needing access to a database is less than the number of fingers on both hands.

Algorithm 3 shows how the watermarking algorithm introduced above can be extended to a fingerprinting algorithm. Although the idea is similar, the fingerprinting algorithm uses a pseudorandom sequence generater G instead of a hash function. We denote the ith number in a sequence generated by G seeded by $t.k|K$ as $G_i(t.k, K)$, where K is the fingerprinting key and $t.k$ is the primary key of tuple t. The fingerprinting algorithm determines which tuples will carry which bit from the fingerprinting code using K and $t.k$. If $G_1(t.k, K) = 0 \ (mod\ p)$ then that tuple is chosen to carry the jth bit from the fingerprinting codeword where $j = G_2(t.k, K) \ (mod\ L)$. To embed a fingerprint codeword of size L, the algorithm first creates $2L$ sets from tuples. Each such set, denoted by S_j, is formed by grouping tuples that satisfy $G_{j+2}(t.a, K) = 0 \ (mod\ p)$, where $t.a$ is the attribute that will be subject to permutations. After deciding which bit of the codeword C is to be embedded, say C_i (ith bit of C) is to be embedded, the attributes from the tuple set $S_{i+C_i L}$ are used for swapping with the original attributes. We skip the analysis of this scheme due to space constraints. However, it is not hard to see that it wouldn't be much different from the watermarking scheme. The added constraint here should be that the expected number of marked tuples in a target detection set size should be big enough to include enough marked tuples to recover all the bits of the codeword with high probability.

Extension to fingerprinting –an alternative approach Although the fingerprinting algorithm described above provides better performance, sometimes a performance loss may be traded for implementation simplicity. For that purpose, we here describe another approach that treats the watermarking algorithm given in algorithm 1 as a black box and embeds fingerprinting codes into the database

Input: Data Set D, G, P, K, C, F
Output: Fingerprinted Data Set D_F
$L \leftarrow LengthOf(C)$
$MarkSet_{1,2,\ldots,2L}, OriginalSet, X \leftarrow \{\}$
$D_F \leftarrow D$
$p \leftarrow \lfloor \frac{1}{P} \rfloor$
foreach *tuple t in* D_F **do**
  if $G_1(t.k, K) = 0(mod\ p)$ **then**
    $i \leftarrow G_2(t.k, K)(mod\ L)$
    if $G_{(2+i+C_iL)}(t.a, K) \neq 0(mod\ pL)$ **then**
      add t into OriginalSet
    end
  end
end
foreach *tuple t in* D_F **do**
  for *i=1 to 2L* **do**
    if $G_{i+2}(t.a, K) = 0\ (mod\ pL)$ **then**
      add t into $Markset_i$
    end
  end
end
foreach *tuple t in OriginalSet* **do**
  $i \leftarrow G_2(t.k, K)(mod\ L)$
  choose a tuple X from $Markset_{i+C_iL}$ that minimizes $F(t.a, X.a)$
  swap(t.a,X.a)
end
return D_F

Algorithm 3: Permutation based Fingerprinting algorithm

by only executing function calls to algorithm 1. The simple algorithm works as follows:

> *Run the watermarking algorithm L times for a codeword C of length L. At the ith ($1 \leq i \leq L$) run of the watermarking algorithm, use K‘ as the watermarking key where $K‘ = K|i|C_i$.*

It is easy to see the number of modifications needed is approximately L times more than watermarking. The worst case running times are; marking takes L times longer and detection takes 2^L times longer compared to watermarking detection alone.

3.2 Insertion Based Watermarking

For certain databases, not just the errors on individual attribute values but also the errors anywhere on a tuple may be detectable. For example, if a tuple consisting of {name, credit card number, expiration date} is modified to embed a watermark, it can be verified for correctness by requesting a charge authorization. Here all the attributes are correlated and any error on the tuple would result in a denied charge authorization revealing which tuples are marked.

Modifying the original records may also not be desirable if it may disrupt the business process. For example, a list of customer addresses which must be notified in case of a product recall cannot tolerate changing any addresses. In such error sensitive databases, the marking cannot be based on any perturbation of the original tuples but may still come from adding extra information to the database. The important requirement here is that the extra tuples should be indistinguishable from the original records.

We developed a technique based on artificial database records[4] that can still be used in the above described situation when previous solutions and permutation based watermarking are not applicable. An artificial record in a data set is a data record that has correct semantics and is qualified for data processing like any other record. For example, a letter to an artificial mailing address will indeed be delivered by the post office and a transaction to an artificial credit card number will go through card processing and appear normal, although behind the scenes they may be processed specially. An artificial credit card number can be used in a purchase transaction, but the data owner and the credit card company are alerted when this credit card number shows up in any transaction. Because the card number is artificial, no legitimate charges should happen, so any attempted use of this artificial number may indicate that the data has been leaked to someone unscrupulous. This is a reliable leak detection mechanism because an efficient market now exists for stolen information, and so leaked information containing credit card data will quickly be sold and used by a well-organized network of criminals, causing transactions using the artificial card numbers. The

[4] The idea of using artificial records was briefly mentioned by Agrawal et. al. in [3] before.

leak source can also be reliably traced if the particular artificial record, or group of records, was provided only to them.

In order to facilitate discovery of leaked data, a trusted third party can be involved, such as the credit card provider aforementioned, both to create the artificial records and monitor the data processing activities associated with the artificial records for leak detection. Since the creation of such records is out of this paper's scope, we assume a set of artificial records that is uniformly distributed over the primary key space is available at hand.
For a given n bit key $K \in \{0,1\}^n$, cryptographic hash function $H(x)$, marking density P and Artificial data set A, algorithm 4 describes how the insertion based watermarking is done.

Input: Data Set D of size S, Artificial Data Set A, P
Output: Watermarked Data Set D_W
$MarkSet \leftarrow \{\}$
while $SizeOf(MarkSet) \leq \frac{SP}{1-P}$ **do**
| insert a randomly chosen tuple from A to $MarkSet$
end
$D_W \leftarrow D \bigcup MarkSet$
return D_W

Algorithm 4: Watermarking algorithm

As you see in algorithm 4, the watermarking procedure is fairly straightforward. For a given marking density P, the algorithm keeps inserting artificial records into the database until the ratio of such records to all records is equal to P.

On the other hand, the detection procedure described in algorithm 5 looks for any artificial tuples in a given database by going through all its tuples. We assume that the records in each company's artificial data set would be unique, so the detection algorithm stops and outputs *true* as soon as it finds a tuple that belongs to the artificial data set. It outputs $false$ if no such tuple is found after going through all the tuples.

Input: Data Set D, Artificial Data Set A
Output: {0,1}
foreach *tuple* t *in* D **do**
| **if** $t \in A$ **then**
| | return 1
| **end**
end
return 0

Algorithm 5: Watermark Detection algorithm

Analysis In this scheme, the value of P determines the ratio of artificial records to all the records in the database. As P increases, the security of the system increases too. In the following formulas, we analyze how the value of P affects certain properties of the database.

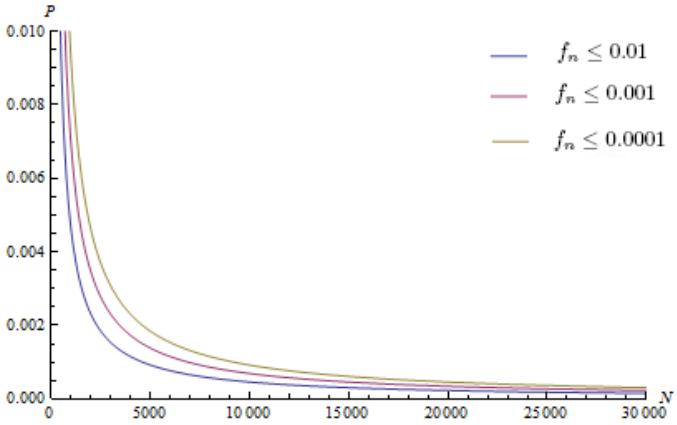

Fig. 3. The expected ratio of inserted virtual tuples by algorithm 4 for different P values

The expected false negative rate when trying to detect one of 10 artificial records in a database of 1000 total records within a leaked subset of size 20 can be precisely calculated with:

$$\frac{990}{1000} \times \frac{989}{999} \times \frac{988}{998} \times \cdots \times \frac{971}{981}$$

The general precise formula is thus:

$$\prod_{1 \leq i \leq N} \frac{(r-m)-i}{r-i}$$

where r is the total database size, m is the number of records that are artificial marking records, and N is the number of records in the leaked subset. Care should be taken when calculating this product to minimize numerical error. Databases of small size like 1000 or fewer records can benefit from this precise formula, but for typical corporate database sizes of 10,000 or more records, a very close and always slightly conservative value can be found using the following useful closed-form approximation. To have not more than f_n false negative probability when the detection algorithm is run on a N or larger sized subset of a database, the value of P should satisfy

$$(1-P)^N < f_n$$

For example, if a false negative rate of less than 0.001 is wanted when a random subset of 10000 or more tuples is given into the detection algorithm, then the P value should satisfy:

$$(1-P)^{10000} < 0.001$$

$$\longrightarrow P \geq 0.0007$$

Figure 3 shows the relation between N and P for different f_n values. In order to set the ratio of virtual tuples to all tuples as P, the number of artificial records needed to mark a database of size S can be calculated as: $SP/(1-P)$

Extension to fingerprinting As in the previous fingerprinting extensions, the fingerprinting codewords are assumed to be binary and the introduced fingerprinting algorithm can be used with any fingerprinting code scheme (collusion-secure or not) that produces binary codewords as long as the length of codewords is a few orders of magnitude smaller than the database size. When fingerprinting a database using codewords of length L, the algorithm first divides the artificial data (tuple) set A into $2L$ subsets. Each subset S_j is formed by grouping tuples that satisfy $H(t.k, K) = j \ (mod\ 2L)$ for $0 \leq j \leq 2L-1$, where $t.k$ is the primary key of tuple $t \in A$. Tuples in each subset are sorted in a non-decreasing order using a sorting function F that takes the primary key of tuples as input and is defined as $F(t.k) = H(t.k)|t.k$, H here is a cryptographic hash function and "|" denotes string concatenation. The watermarking algorithm (Algorithm 4) is run L times to fingerprint a codeword C of length L. At the ith run, the watermarking algorithm only uses the virtual tuples from subset S_{i+C_iL}, where C_i is the value of the ith least significant bit of C, and the artificial tuples are used in the order they reside in the sorted sets.

We skip the analysis due to space constraints but it is not hard to see the number of virtual tuples needed would be L times of the number of tuples needed just for watermarking (for similar error rates at detecting each bit of the fingerprint). However, the error rate using the fingerprinting algorithm to successfully recover the whole codeword would be larger than the error rate of the watermarking algorithm to successfully detect the mark. This is due to the fact that not being able detect even one bit out of L is still considered as unsuccessful and the probabilities of not being able to detect each bit add up.

3.3 Encryption Based Approach

In business process outsourcing, sometimes the service provider doesn't need to see all the attributes in a database to be able to use it[5]. A service provider may just play a transitive role in the process and the real values of those attributes are eventually processed by another service provider. Common examples for this kind of data are social security numbers (SSNs) or credit card numbers (CCNs). To better explain the data flow here, assume a company A outsources its employee background check process to company B. This process usually includes obtaining

[5] To stay within the scope, we describe possible usages of this scheme in the domain of business outsourcing, but we want to note that the described scheme is equally applicable to prevent leaks from in-house storage as long as the aim of storing sensitive data is to later take it to another entity for processing (e.g., storing credit card numbers to later take them to a credit card transaction clearinghouse).

the credit history for each employee so the SSN of each employee should be in the database that is handed to company B to execute the process. However, the only thing company B will do with those SSNs is to take them to a credit bureau C and obtain the corresponding credit history reports. Company B doesn't need to see the actual numbers if it can still acquire the corresponding reports and it obviously doesn't need the SSNs to evaluate the reports after obtaining them. In this scenario, the numbers may be encrypted and kept hidden from company B as long as they can be decrypted and processed by entity C and can be linked back to the original numbers by company A.

The idea described above can easily be implemented using public key cryptography and it would keep the SSNs confidential if they are encrypted using company C's public key. However, it doesn't prevent them from being used by unauthorized parties in case of a leak. This problem can easily be addressed by company A specifying who it authorizes to use the data inside the encrypted blob and company C verifying the identity of Company B before responding to any queries.

It is important to note that whenever this scheme is applicable, it provides much better security than watermarking or fingerprinting as it is a preventative security measure unlike the other two. Luckily, the highly sensitive data in the business world (e.g., SSN, CCN, employee or bank account numbers, etc...) tends be governed by only a few well defined authorities to keep the risk manageable, and this makes it easier to take advantage of the already available public key infrastructure when using this scheme. However, encryption wouldn't work if Company B needs the sensitive information in plain text to be able to properly execute the outsourced process and so the application domain is somewhat limited.

Method definitions and protocol details We use A, B and C in the same way they are used in the previously given example, that is, A to denote the database owner, B to denote the service provider and C to denote the company that will eventually process the data. We use $E(K, M)$ to denote symmetric encryption of message M using the encryption key K while $E_X(M)$ to denote the asymmetric encryption of M under the public key of entity X. We assume entities B and C have X.509 public key certificates obtained from a trusted certification authority.
Encryption: If column m of database D contains the sensitive information, A encrypts all the corresponding column of the database using a n bit symmetric key K and creates D'. After this operation, any attribute a residing in that column is in the form of $E(K, a)$ in D'. A then creates a token to be given to B together with D'. The token is created as follows:

$$\text{token} = E_C(K|A|B|expiration\_date|beginning\_date|notification\_request)$$

Process execution: When B needs the encrypted attributes to be processed by C as part of the business process it is handling, it always sends the token it received from A together with the encrypted data.

Decryption and authentication: Whenever C receives a request to process encrypted data, before honoring the request it first decrypts the accompanying token and does the the following checks

- Acquire the public key certificate for B and verify its authenticity.
- Using a challenge-response protocol, verify that the party sending the request is the real owner of the acquired certificate (has the corresponding private key).
- Check that the current date is between the $beginning\_date$ and the $expiration\_date$

If all the above checks are successful, C then honors the request of B and notifies A if the $notification\_request$ flag in the token was set.

4 Discussion

The value placed upon the confidentiality and integrity of corporate databases varies immensely. Preserving the confidentiality of a small customer list containing only addresses may be worth several thousands of dollars. Preserving the confidentiality and integrity of a large customer list with historical sales information, complaint history, and other data could easily be valued by the corporation in the millions of dollars. Thus, techniques like those we and others are developing are needed in order to confidently allow more sensitive data to be securely shared with outsourcing providers, which enables the business to optimize its processes and reduce its costs.

The presented watermarking techniques are practical for many types of corporate databases, including those for CRM (Customer Relationship Management), Product or Portfolio Management, and many custom databases. If the business process being delivered by the outsourcing provider is tolerant to a small percentage of errors, then the permutation-based technique can be used to automatically create different watermarked databases which can then be given to different providers. If, on the other hand, the business process being outsourced is not tolerant of errors in the source data, but is tolerant of the addition of a small percentage of artificial data records, then the insertion-based technique can be used. In addition to watermarking, the insertion-based technique has a nice feature that inserted artificial tuples can be used to monitor the provider's service quality and also to detect any leaks as soon as data is used by unauthorized entities.

In the insertion-based watermarking technique, we assume a set of artificial records is available. The generation process for artificial records depends heavily on the data schema. In some cases, parts of different existing records can be automatically combined to create the artificial records. For example, a first name from a random record can be paired with a last name from a different random record to create an artificial full name record. In other cases, a person should be involved in creating the set of artificial records to ensure that they are not easily detectable, and possibly also to enable verification of the outsourced business

process. For example some home mailing addresses of employees in the department could be used, possibly with different names as well. There are many cases when preserving the confidentiality of the outsourced data is worth the time to create these artificial records. However, a detailed treatment of the generation process for artificial records is beyond the scope of this paper.

In terms of the accuracy and effectiveness of our marking algorithms, any data leaks detected can be traced to a particular provider, with an arbitrarily low chance of a false positive leak detection using the permutation technique, or with no chance of false positive leak detection using the insertion technique. Knowing that they would probably be caught provides the outsourcing provider and its personnel with a strong deterrent against leaking the data. Typically a corporation may just end the relationship with a provider which leaks its data. But there is opportunity to create a business which watermarks and fingerprints data and maintains a strong chain of evidence so that providers which leak data could be held accountable in court. Such a business would need to ensure that it doesn't have net incentive to leak the marked data and then frame an outsourcing provider[6], since it would have the ability to do so.

Observing different BPO scenarios encouraged us to specify a protocol that can be used to prevent data leakage in certain scenarios. As discussed in section 3.3, the encryption-based technique has quite a limited use case scenario. However, those limited scenarios are usually the ones that deal with the most sensitive data like credit card, social security or bank account numbers, where preventative security measures are highly preferable to reactive ones. We present this technique complementary to the other two as in certain cases, it may still be used when any available watermarking would fail to provide the required security or be inapplicable.

5 Conclusion

In this paper, we introduced two new techniques that can be used to watermark relational databases. Unlike previous solutions, which are limited to be applicable only on numerical relational data, our methods can accommodate databases with no numerical attributes. For databases with numerical data, the methods introduced in this paper are complementary to the previous work as our methods do not change individual attribute values and can remain secure in many cases where the others would fail. We also showed how our watermarking techniques can be extended to fingerprinting algorithms that can accommodate collusion secure binary codewords.

Our insertion-based technique can provide more than watermarking if the artificial records to be used are chosen wisely. In many cases those artificial tuples can be used as honeypots to quickly detect any unauthorized use as well as to monitor and evaluate the quality of the received service.

[6] This problem is well known in the literature and solutions not allowing the content owner to frame a copy holder are known as asymmetric fingerprinting schemes.

Complementary to the watermarking techniques, we also described how the existing public key infrastructure can be utilized for better data protection. The introduced encryption-based technique provides strong and preventative security measures against data leakage in BPO when relational data containing sensitive information needs to be stored in transitive entities.

References

1. R. Agrawal, P. Haas, and J. Kiernan. A system for watermarking relational databases. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 674–674, 2003.
2. R. Agrawal, P. Haas, and J. Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(2):157–169, 2003.
3. R. Agrawal and J. Kiernan. Watermarking relational databases. *Proceedings of the 28th international conference on Very Large Data Bases-Volume 28*, pages 155–166, 2002.
4. M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. Triezenberg. Natural Language Watermarking and Tamperproofing. *Information Hiding: 5th International Workshop, IH 2002, Noordwijkerhout, the Netherlands, October 7-9, 2002: Revised Papers*, 2003.
5. C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp. Dynamic path-based software watermarking. *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 107–118, 2004.
6. C. Collberg and C. Thomborson. Software watermarking: models and dynamic embeddings. *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 311–324, 1999.
7. I. Cox, J. Killian, T. Leighton, and T. Shamoon. Secure spread spectrum communication for multimedia. *IEEE Trans. Image Processing*, 6(1):2, 1997.
8. I. Cox, M. Miller, J. Bloom, and C. Honsinger. Digital Watermarking. *Journal of Electronic Imaging*, 11:414, 2002.
9. K. Fukushima and K. Sakurai. A software fingerprinting scheme for java using classfiles obfuscation. *LNCS*, 2908:303–316, 2003.
10. D. Gross-Amblard. Query-preserving watermarking of relational databases and XML documents. *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 191–201, 2003.
11. F. Guo, J. Wang, Z. Zhang, X. Ye, and D. Li. An Improved Algorithm to Watermark Numeric Relational Data. *Proceedings of WISA*, pages 138–149, 2005.
12. M. Huang, J. Cao, Z. Peng, and Y. Fang. A new watermark mechanism for relational data. *Computer and Information Technology, 2004. CIT'04. The Fourth International Conference on*, pages 946–950, 2004.
13. N. Johnson, Z. Duric, S. Jajodia, and N. Memon. Information Hiding: Steganography and WatermarkingAttacks and Countermeasures. *Journal of Electronic Imaging*, 10:825, 2001.
14. S. Katzenbeisser and F. Petitolas. Information Hiding Techniques for Steganography and Digital Watermaking. *EDPACS*, 28(6):1–2, 2000.
15. Y. Li, V. Swarup, and S. Jajodia. Fingerprinting Relational Databases: Schemes and Specialties. *IEEE Transactions On Dependable and Secure Computing*, pages 34–45, 2005.

16. S. Liu, S. Wang, R. Deng, and W. Shao. A Block Oriented Fingerprinting Scheme in Relational Database. *Proc. Seventh Ann. Int'l Conf. Information Security and Cryptology (ICISC)*, 2004.
17. R. Sion. Proving ownership over categorical data. *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 584–595, March-2 April 2004.
18. R. Sion, M. Atallah, and S. Prabhakar. Rights Protection for Relational Data. *IEEE Transactions on Knowledge and Data Engineering*, pages 1509–1525, 2004.
19. H. Steinberg and S. Gordon. Software fingerprinting and branding. US Patent 6,574,732, 2003.
20. M. Topkara, G. Riccardi, D. Hakkani-Tuer, and M. Atallah. Natural language watermarking: challenges in building a practical system. *Proceedings of SPIE*, 6072:106–117, 2006.
21. United States Postal service. Address verification tools. `http://www.usps.com/business/addressverification/welcome.htm`.
22. R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. *4th International Information Hiding Workshop*, 2001.
23. Y. ZHANG, X. Niu, and D. Zhao. A Method of Protecting Relational Databases Copyright with Cloud Watermark. *International Journal of Information Technology*, 1(4):206–210, 2004.