Development of Data leakage Detection Using Data Allocation Strategies

Polisetty Sevani Kumari[1], Kavidi Venkata Mutyalu[2]

[1] M.Tech, CSE, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem
W.G.Dt., A.P. India.
[2]Asst. Professor, Dept of IT, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem.
W.G.Dt. A.P., India

Abstract— **A data distributor has given sensitive data to a set of supposedly trusted agents (third parties). Some of the data are leaked and found in an unauthorized place. The distributor must assess the likelihood that the leaked data came from one or more agents, as opposed to having been independently gathered by other means. We propose data allocation strategies (across the agents) that improve the probability of identifying leakages. These methods do not rely on alterations of the released data (e.g., watermarks). In some cases, we can also inject "realistic but fake" data records to further improve our chances of detecting leakage and identifying the guilty party.**
In previously only one agent is going to leak the data but by using allocation strategies we are going to create multiple agents. this project is possible to show in stand alone system, but now we are going to show the result dynamically using MVC architecture.

I. INTRODUCTION

In the course of doing business, sometimes sensitive data must be handed over to supposedly trusted third parties. For example, a company may have partnerships with other companies that require sharing customer data. Another enterprise may outsource its data processing, so data must be given to various other companies. Our goal is to detect when the distributor's sensitive data have been leaked by agents, and if possible to identify the agent that leaked the data. Perturbation is a very useful technique where the data are modified and made "less sensitive" before being handed to agents For example, one can replace exact values by ranges, or one can add random noise to certain attributes.
Traditionally, leakage detection is handled by watermarking. We annunciate the need for watermarking database relations to deter their piracy, identify the unique characteristics of relational data which pose new challenges for watermarking, and provide desirable properties of watermarking system for relational data. A watermark can be applied to any database relation having attributes which are such that changes in a few of their values do not affect the applications. Watermarking means a unique code is embedded in each distributed copy If that copy is later discovered in the hands of an unauthorized party, the leaker can be identified. Furthermore, watermarks can sometimes be destroyed if the data recipient is malicious.

In this paper, we study unobtrusive techniques for detecting leakage of a set of objects or records. Specifically, we study the following scenario: After giving a set of objects to agents, the distributor discovers some of those same objects in an unauthorized place. At this point, the distributor can assess the likelihood that the leaked data came from one or more agents, as opposed to having been independently gathered by other means. Using an analogy with cookies\ stolen from a cookie jar, if we catch Freddie with a single cookie, he can argue that a friend gave him the cookie. But if we catch Freddie with five cookies, it will be much harder for him to argue that his hands were not in the cookie jar. If \ the distributor sees "enough evidence" that an agent leaked data, he may stop doing business with him, or may initiate\ legal proceedings.

In this paper, we develop a model for assessing the "guilt" of agents. We also present algorithms for distributing objects to agents, in a way that improves our chances of identifying a leaker. Finally, we also consider the option of adding "fake" objects to the distributed set. Such objects do not correspond to real entities but appear realistic to the agents. In a sense, the fake objects act as a type of watermark for the entire set, without modifying any individual members.

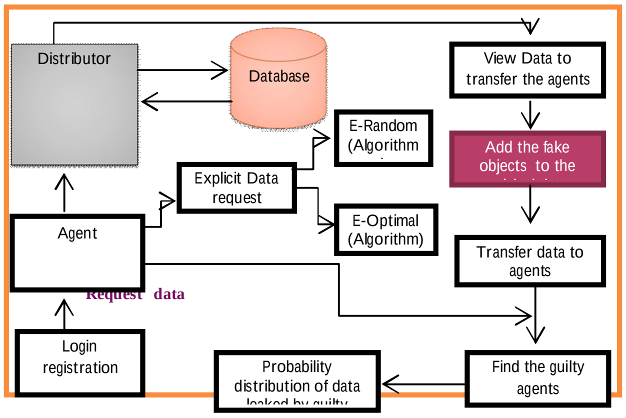


Fig1.System Architecture

II. DATA ALLOCATION PROBLEM

A. *Fake objects*

The distributor may be able to add fake objects to the distributed data in order to improve his effectiveness in detecting guilty agents. However, fake objects may impact the correctness of what agents do, so they may not always be allowable.
The idea of perturbing data to detect leakage is not new, e.g., [1]. However, in most cases, individual objects are Perturbed, e.g., by adding random noise to sensitive salaries,or adding fake elements.

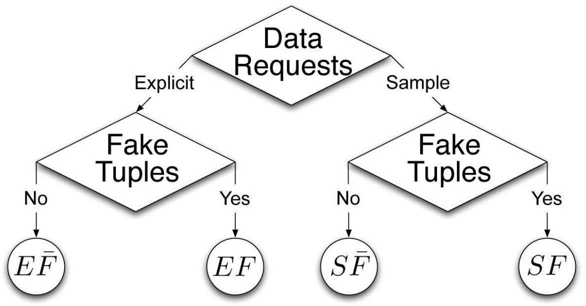
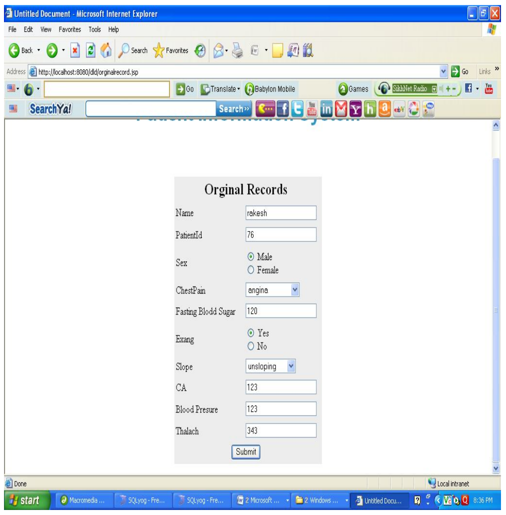


Fig2. Leakage problem instances.

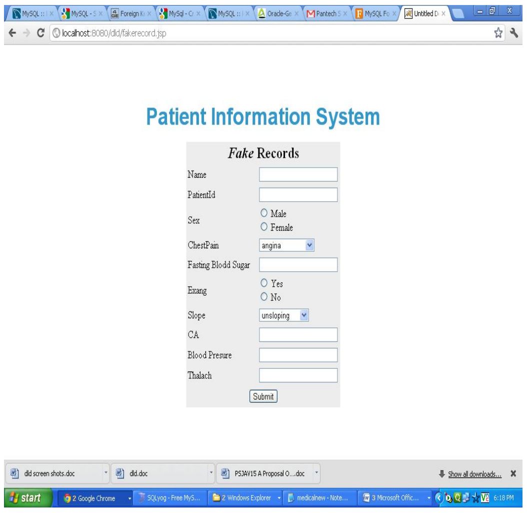
In some applications, fake objects may cause fewer problems that perturbing real objects.
Creation. The creation of fake but real-looking objects is a nontrivial problem whose thorough investigation is beyond the scope of this paper. Here, we model the creation of a fake object for agent Ui as a black box function CREATEFAKEOBJECT (Ri, Fi, condi) that takes as input the set of all objects Ri, the subset of fake objects Fi that Ui has received so far, and condi, and returns a new fake object. This function needs condi to produce a valid object that satisfies Ui's condition. Set Ri is needed as input so that the created fake object is not only valid but also indistinguishable from other real objects. For example, the creation function of a fake payroll record that includes an employee rank and a salary attribute may take into account the distribution of employee ranks, the distribution of salaries, as well as the correlation between the two attributes. Ensuring that key statistics do not change by the introduction of fake objects is important if the agents will be using such statistics in their work. Finally, function CREATEFAKEOBJECT () has to be aware of the fake objects Fi added so far, again to ensure proper statistics. The distributor can also use function CREATEFAKEOBJECT () When it wants to send the same fake object to a set of agents. In this case, the function arguments are the union of the Ri and Fi tables, respectively, and the intersection of the conditions condi s.

Although we do not deal with the implementation of CREATEFAKEOBJECT (), we note that there are two main design options. The function can either produce a fake object on demand every time it is called or it can return an appropriate object from a pool of objects created in advance.
We are using the following strategies to add the fake object to finding guilty agent,


Orginal Records

Name	rakesh
PatientId	76
Sex	[✓ Male] Female
ChestPain	angina
Fasting Blodd Sugar	120
Exang	[✓ Yes] No
Slope	unsloping
CA	123
Blood Presure	123
Thalach	343

Fig3. Adding the Original records


Patient Information System

Fake Records

Name	
PatientId	
Sex	Male Female
ChestPain	angina
Fasting Blodd Sugar	
Exang	Yes No
Slope	unsloping
CA	
Blood Presure	
Thalach	

Fig4. Adding the fake objects

B. optimization problem

The distributor's data allocation to agents has one constraint and one objective. The distributor's constraint is to satisfy agents' requests, by providing them with the number of objects they request or with all available objects that satisfy their conditions. His objective is to be able to detect an agent who leaks any portion of his data. We consider the constraint as strict. The distributor may not deny serving an agent request and may not provide agents with different perturbed versions of the same objects as in [1]. We consider fake object distribution as the only possible constraint relaxation. Our detection objective is ideal and intractable. Detection would be assured only if the distributor gave no data object to any agent. We use instead the following objective: maximize the chances of detecting a guilty agent that leaks all his data objects. We now introduce some notation to state formally the distributor's objective. Recall that Pr {Gj/S= Ri} or simply Pr {Gj/Ri}is the probability that agent Uj is guilty if the distributor discovers a leaked table S that contains all Ri objects. We define the difference functions Δ(i, j) as

Δ(i, j)=Pr{Gi /Ri}-Pr{Gj/Ri} i, j=1,…..n. …………(1)

Note that differences _ have nonnegative values: given that Set Ri contains all the leaked objects, agent Ui is at least as likely to be guilty as any other agent. Difference Δ(i,j) is positive for any agent Uj, whose set Rj does not contain all data of S. It is zero if Ri$\subseteq$ Rj. In this case, the distributor will consider both agents Ui and Uj equally guilty since they have both received all the leaked objects. The larger a Δ (i, j) value is, the easier it is to identify Ui as the leaking agent. Thus, we want to distribute data so that _ values are large. **Problem Definition**. Let the distributor have data requests from n agents. The distributor wants to give tables R1, ...,Rn to agents U1, . . . , Un, respectively, so that . he satisfies agents' requests, and . he maximizes the guilt probability differences Δ(I,j) for all i ,j=1and i$\neq$j. Assuming that the Ri sets satisfy the agents' requests, we can express the problem as a multicriterion optimization problem:

maximize (….,Δ(i ,j),..) i$\neq$j. ……….. (2)

If the optimization problem has an optimal solution, it means that there exists an allocation D*= (R1*,…, Rn*) such that any other feasible allocation yields D*=(R1*,…,Rn*} yields Δ(i,j)$\geq\Delta$*(i,j) for all i; j. This means that allocation Tj* allows the distributor to discern any guilty agent with higher confidence than any other allocation, since it maximizes the probability Pr {Gi/Ri} with respect to any other probability Pr {Gi/Rj} with j $\neq$i. Even if there is no optimal allocation D*, a multicriterion problem has Pareto optimal allocations.

C. Objective Approximation

We can approximate the objective of (2) with (3) that does not depend on agents' guilt probabilities, and therefore, on p:

Maximise (..,|Ri$\cap$Rj|/|Ri|,….) i $\neq$j. …………(3)
(over R1,..Rn)

This approximation is valid if minimizing the relative overlap |Ri$\cap$Rj|/|Ri| maximizes Δ (i ,j).

The intuitive argument for this approximation is that the fewer leaked objects set Rj contains, the less guilty agent Uj will appear compared to Ui (since S = Ri). In Fig. 1, we see that if S = R1, the difference Pr {G1/S}-Pr {G2/S} decreases as the relative overlap|jR2$\cap$S| /|S| increases.

Theorem 1: If a distribution D= {R1,….., Rn} that satisfies agents' requests minimizes {Ri$\cap$Rj|/|Ri| and |Vt|=|Vt'| for all t,t'$\in$T, then D Maximizes(i,j).The approximate optimization problem has still multiple criteria and it can yield either an optimal or multiple Pareto optimal solutions. Pareto optimal solutions let us detect a guilty agent Ui with high confidence, at the expense of an inability to detect some other guilty agent or agents. Since the distributor has no a priori information for the agents' intention to leak their data, he has no reason to bias the object allocation against a particular agent. Therefore, we can scalarize the problem objective by assigning the same weights to all vector objectives.

Maximize $\sum$i=1 1/|Ri|$\sum$j=1|Ri $\cap$ Rj| …….. (4a)
(Over R1… Rn)

Maximize max |Ri $\cap$ Rj| /|Ri| ……… (4b)
(Over R1… Rn)

Both scalar optimization problems yield the optimal solution of the problem of (3), if such solution exists. If there is no global optimal solution, the sum-objective yields the Pareto optimal solution that allows the distributor to detect the guilty agent, on average (over all different agents), with higher confidence than any other distribution. The max-objective yields the solution that guarantees that the distributor will detect the guilty agent with certain confidence in the worst case. Such guarantee may adversely impact the average performance of the distribution.

III. ALLOCATION STRATEGIES

In this section, we describe allocation strategies that solve exactly or approximately the scalar versions of (3) for the different instances presented in Fig. 2. We resort to approximate solutions in cases where it is inefficient to solve accurately the optimization problem. In Section (a), we deal with problems with explicit data requests, and in Section (b), with problems with sample data requests. proofs of theorems that are stated in the following sections are available in [14].

A. Explicit Data Requests

In problems of class EF, the distributor is not allowed to add fake objects to the distributed data. So, the data allocation is fully defined by the agents' data requests. Therefore, there is nothing to optimize. In EF problems, objective values are initialized by agents' data requests. Say, for example, that T= {t1, t2} and there are two agents with explicit data requests such\ that R1= {t1, t2} and R2 = {t1|. The value of the sum objective is in this case

Σi=1 1/|Ri|Σj=1|Ri $\cap$ Rj|=1/2+1/1=1.5.

The distributor cannot remove or alter the R1 or R2 data to decrease the overlap R1$\cap$R2. However, say that the distributor can create one fake object (B= 1) and both agents can receive one fake object (b1= b2= 1). In this case, the distributor can add one fake object to either R1 or R2 to increase the corresponding denominator of the summation term. Assume that the distributor creates a fake object f and he gives it to agent R1. Agent U1 has now R1= {t1, t2, f} and F1={f} and the value of the sum-objective decreases to 1/3+1/1=1.33<1.5.

```
Algorithm 1. Allocation for Explicit Data Requests (EF)
Input: R1. . . Rn, cond1; . . . ; condn, b1,. . . , bn, B
Output: R1. . . Rn, F1. . . Fn
1: R ←ϕ     Agents that can receive fake objects
2: for i =1… n do
```

```
3: if bi > 0 then
4: R←R∪ {i}
5: Fi ←ϕ     ;
6: while B > 0 do
7: i ← SELECTAGENT(R, R1. . .Rn)
8: f ← CREATEFAKEOBJECT (Ri, Fi, condi)
9: Ri ← Ri ∪ {f}
10: Fi ← Fi ∪{f}
11: bi ← bi _ 1
12: if bi = 0then
13: R← R/ {Ri}
14: B← B _ 1
```

Algorithm 2. Agent Selection for e-random

```
1: function SELECTAGENT (R, R1; . . ., Rn)
2: i← select at random an agent from R
3: return i
```

In lines 1-5, Algorithm 1 finds agents that are eligible to receiving fake objects in O(n) time. Then, in the main loop in lines 6-14, the algorithm creates one fake object in every iteration and allocates it to random agent. The main loop takes O (B) time. Hence, the running time of the algorithm is O(n+ B). If B ≥Σn i=1 bi, the algorithm minimizes every term of the objective summation by adding the maximum number bi of fake objects to every set Ri, yielding the optimal solution. Otherwise, if B < Σi=1 bi (as in our example where B=1 < b1+b2 =2), the algorithm just selects at random the agents that are provided with fake objects. We return back to our example and see how the objective would change if the distributor adds fake object f to R2 instead of R1. In this case, the sum-objective would be 1/ 2+1/2=1<1.33. The reason why we got a greater improvement is that the addition of a fake object to R2 has greater impact on the corresponding summation terms, since 1/|R1|-1/|R1|+1=1/6<1/|R2|-1/|R2|+1=1/2.

The left-hand side of the inequality corresponds to the objective improvement after the addition of a fake object to R1 and the right-hand side to R2.

Algorithm 3. Agent Selection for e-optimal

```
1: function SELECTAGENT (R,R1; . . .;Rn)
2: i← argmax (1/|Ri′|-1/|Ri′|+1)Σ|Ri′∩Rj|
3: return i
```

Algorithm 3 makes a greedy choice by selecting the agent that will yield the greatest improvement in the sum objective the cost of this greedy choice is O (n2) in every iteration. The overall running time of e-optimal is O (n+n2B) =O (n2B). Theorem 2 shows that this greedy approach finds an optimal distribution with respect to both optimization objectives defined in (4).

Theorem 2. Algorithm e-optimal yields an object allocation that minimizes both sum- and max-objective in problem instances of class EF.

B. Sample Data Requests

With sample data requests, each agent Ui may receive any T subset out of (|T|) different ones. Hence, there are ∏i=1(|T|) different object allocations. In every allocation, the distributor can permute T objects and keep the same chances of guilty agent detection. The reason is that the guilt probability depends only on which agents have received the leaked objects and not on the identity of the leaked objects. Therefore, from the distributor's perspective, different allocations. The distributor's problem is to pick one out so that he optimizes his objective. We formulate the problem as a nonconvex QIP that is NP-hard.

Note that the distributor can increase the number of possible allocations by adding fake objects (and increasing |T|) but the problem is essentially the same. So, in the rest of this section, we will only deal with problems of class SF, but our algorithms are applicable to SF problems as well.

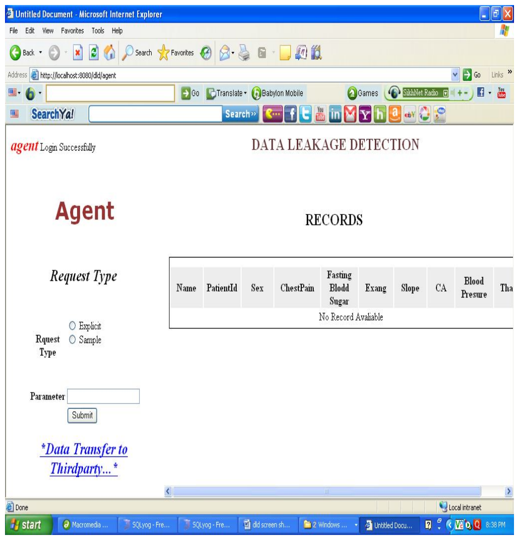


Fig5. Requesting the data

C. Random

An object allocation that satisfies requests and ignores the distributor's objective is to give each agent Ui a randomly selected subset of T of size mi. We denote this algorithm by S-random and we use it as our baseline. We present S-random in two parts: Algorithm 4 is a general allocation algorithm that is used by other algorithms in this section. In line 6 of Algorithm 4, there is a call to function SELECTOBJECT () whose implementation differentiates algorithms that rely on Algorithm 4. Algorithm 5 shows function SELECTOBJECT () for s-random.

Algorithm 4. Allocation for Sample Data Requests (SF)

Input: m1, . . ,mn, |T| . Assuming mi <|T|

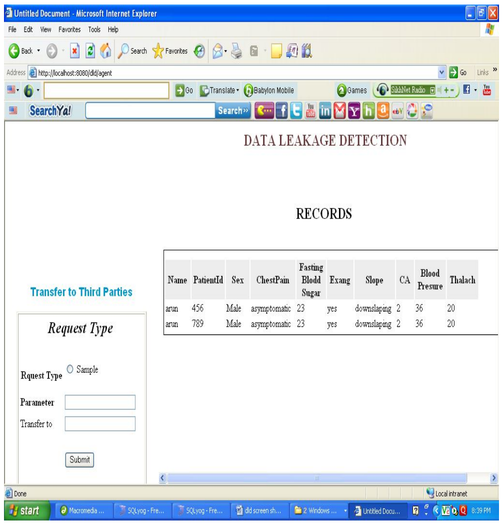
Output: R1, . . .,Rn

```
1: a← 0|T|.   a[k]:number of agents who have received
object tk
2: R1 ←ϕ. . . Rn ←ϕ
3: remaining ←Σi=1 mi
4: while remaining > 0 do
```

```
5: for all i = 1. . . n: |Ri| <mi do
6: k← SELECTOBJECT (i, Ri)   May also use additional parameters
7: Ri ← Ri ∪{tk}
8: a[k] ← a[k] + 1
9: remaining ←remaining _ 1
```

Algorithm 5. Object Selection for s-random

```
1: function SELECTOBJECT (i, Ri)
2: k ←select at random an element from set {K′/tk∉Ri}
3: return k
```



DATA LEAKAGE DETECTION

RECORDS

Name	PatientId	Sex	ChestPain	Fasting Blodd Sugar	Exang	Slope	CA	Blood Presure	Thalach
arun	456	Male	asymptomatic	23	yes	downslaping	2	36	20
arun	789	Male	asymptomatic	23	yes	downslaping	2	36	20

Transfer to Third Parties

Request Type

Rquest Type ○ Sample

Parameter

Transfer to

Submit

Untitled Document - Microsoft Internet Explorer
http://localhost:8080/dld/agent
Done
Local intranet
8:39 PM

Fig6. Transferring the object to third person

D. *Overlap Minimization*

In the last example, the distributor can minimize both objectives by allocating distinct sets to all three agents. Such an optimal allocation is possible, since agents request in total fewer objects than the distributor has. We can achieve such an allocation by using Algorithm 4 and SELECTOBJECT () of Algorithm 6. We denote the resulting algorithm by s-overlap. Using Algorithm 6, in each iteration of Algorithm 4, we provide agent Ui with an object that has been given to the smallest number of agents. So, if agents ask for fewer objects than |T|, Algorithm 6 will return in every iteration an object that no agent has received so far. Thus, every agent will receive a data set with objects that no other agent has.

Algorithm 6. Object Selection for s-overlap

```
1: function SELECTOBJECT (i, Ri, a)
2: K←{k| k= argmin a[k′]}
3: k← select at random an element from set {k′|k′∈k∧tk′∉Ri}
4: return k
```

The running time of Algorithm 6 is O (1) if we keep in memory the set {k| k= argmin k′ a[k′]}. This set contains initially all indices {1. . . |T|}, since a[k′]} for all k =1, .. |T|. After every Algorithm 4 main loop iteration, We remove from this set the index of the object that we give to an agent. After |T| iterations, this set becomes empty and we have to reset it again to {1,..,|T|}, since at this point, a[k]= 1 for all k = 1, . . . ; |T|. The total running time of Algorithm s-random is thus O (Σi=1 mi).

Let M =Σi=1 mi. If M ≤|T|, algorithm s-overlap yields disjoint data sets and is optimal for both objectives (9a) and (9b). If M >|T|, it can be shown that algorithm s-random yield an object sharing distribution such that:

a[k]={ M÷ |T|+ 1 for M mod|T| entries of vector a;
M÷ |T| for the rest:

Theorem 3. In general, Algorithm s-overlap does not minimize sum-objective. However, s-overlap does minimize the sum of overlaps. To illustrate that s-overlap does not minimize the sum objective, assume that set T has four objects and there are four agents requesting samples with sizes m1=m2= 2 and m3 = m4= 1. A possible data allocation from s-overlap is R1= {t1, t2}, R2= {t3, t4}; R3= {t1}, R4 = {t3}. ……….. (5)

Allocation (10) yields:

Σ1/|Ri|Σ|Ri∩Rj|=1/2+1/2+1/1+1/1=3

With this allocation, we see that if agent U3 leaks his data, We will equally suspect agents U1 and U3. Moreover, if agent U1 leaks his data, we will suspect U3 with high probability, since he has half of the leaked data. The situation is similar for agents U2 and U4. However, the following object allocation R1= {t1,t2}, R2 ={t1, t2}, R3={t3},R4={t4} ………. (6)

Yields a sum-objective equal to 2/ 2+2/2+0+0=2<3 which shows that the first allocation is not optimal. With this allocation, we will equally suspect agents U1 and U2 if either of them leaks his data. However, if either U3 or U4 leaks his data, we will detect him with high confidence. Hence, with the second allocation we have, on average, better chances of detecting a guilty agent.

E. *Approximate Sum-Objective Minimization*

The last example showed that we can minimize the sum objective, and therefore, increase the chances of detecting a guilty agent, on average, by providing agents who have mall requests with the objects shared among the fewest agents. This way, we improve our chances of detecting guilty agents with small data requests, at the expense of reducing our chances of detecting guilty agents with large data requests. However, this expense is small, since the probability to detect a guilty agent with many objects is less affected by the fact that other agents have also received his data. We provide an algorithm that implements this intuition and we denote it by s-sum. Although we evaluate this algorithm in Section 8, we do not present the pseudo code here due to the space limitations

SELECTOBJECT () procedure in Algorithm 7. We denote the new algorithm by s-max. In this algorithm, we allocate to an agent the object that yields the minimum increase of the maximum relative overlap among any pair of agents. If we apply s-max to the example above, after the first five main loop iterations in Algorithm 4, the Ri sets are:
R1={t1, t2}; R2 ={t2,},R3 ={t3}; and R4 ={t4}:
In the next iteration, function SELECTOBJECT () must decide which object to allocate to agent U2. We see that only objects t3 and t4 are good candidates, since allocating t1 to U2 will yield a full overlap of R1 and R2. Function SELECTOBJECT () of s-max returns indeed t3 or t4.

```
Algorithm 7. Object Selection for s-max
1: function SELECTOBJECT (i, R1, . . .,Rn,m1, . . .,mn)
2: min_ overlap←1        the minimum out of the maximum
relative overlaps that the allocations of different objects to
Ui yield
3: for k ∈ {k′|tk′∉ Ri} do
4: max_ rel_ ov←0. the maximum relative overlap between
Ri and any set Rj that the allocation of tk to Ui yields
5: for all j = 1… n: j ≠ i and tk ∉ Rj do
6: abs_ ov←|Ri ∩ Rj |+ 1
7: rel_ ov ←abs_ ov/min (mi, mj)
8: max _rel_ ov← Max (max_ rel_ ov, rel_ ov)
9: if max_ rel_ ov ≤  min_ overlap then
10: min_ overlap ←max_ rel_ ov
11: ret _k← k
12: return ret_ k
```

IV. BENEFITS

In Section 7, we presented algorithms to optimize the Problem of (3) that is an approximation to the original optimization problem of (2). In this section, we evaluate the Presented algorithms with respect to the original problem. In this way, we measure not only the algorithm performance, but also we implicitly evaluate how effective the approximation is.
The objectives in (2) are the Δdifference functions. Note that there are n(n-1)objectives, since for each agent Ui, there are n-1 differencesΔ(i, j)for j=1, . . . , n and j ≠ i.
We evaluate a given allocation with the following objective scalarizations as metrics:

$\Delta := \Sigma i\ ,\ j=1,..n,\ i\neq j\ \Delta(i,j)/n(n-1)$ ……….(7a)

$\min\Delta := \min i\ ,\ j=1,..n\ ,i\neq j\ \Delta\ (i,j).$ ………(7b)

Metric Δ is the average of Δ (i, j) values for a given allocation and it shows how successful the guilt detection is, On average, for this allocation. For example, ifΔ=0:4, then, on average, the probability Pr{Gi/Ri} for the actual guilty agent will be 0.4 higher than the probabilities of nonguilty agents. Note that this scalar version of the original problem Objective is analogous to the sum-objective scalarization of the problem of (3). Hence, we expect that an algorithm that is designed to minimize the sum-objective will maximize _.

Metric min Δ is the minimumΔ (i, j) value and it corresponds to the case where agent Ui has leaked his data and both Ui and another agent Uj have very similar guilt probabilities. If min Δ is small, then we will be unable to Identify Ui as the leaker, versus Uj. If minΔ is large, say, 0.4, then no matter which agent leaks his data, the probability that he is guilty will be 0.4 higher than any other nonguilty agent. This metric is analogous to the max-objective scalarization of the approximate optimization problem. The values for these metrics that are considered acceptable will of course depend on the application. In particular, they depend on what might be considered high confidence that an agent is guilty. For instance, say that Pr {Gi/Ri} = 0.9 is enough to arouse our suspicion that agent Ui leaked data.

Furthermore, say that the difference between Pr {Gi/Ri} and any other Pr{Gj/Ri} is at least 0.3. In other words, the guilty agent is (0.9 – 0.6)/0.6∗ 100% = 50% more likely to be guilty compared to the other agents. In this case, we may be willing to take action against Ui (e.g., stop doing business with him, or prosecute him). In the rest of this section, we will use value 0.3 as an example of what might be desired in values. To calculate the guilt probabilities and differences, we use throughout this section p =0.5. Although not reported here, we experimented with other p values and observed that the relative performance of our algorithms and our main conclusions do not change. If p approaches to 0, it becomes easier to find guilty agents and algorithm performance converges. On the other hand, if p approaches 1, the relative differences among algorithms grow since more evidence is needed to find an agent guilty.

V. SAMPLE IMPLEMENTATION CODE

```
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.http.HttpSession.*;
import java.io.*;
import java.util.*;
import javax.sql.*;
public class FakeRegistration extends HttpServlet
{
 HttpSession hs;
 PrintStream ps,ps1;
Connection con;
PreparedStatement st;
ResultSet rs;
 // String str=null;
String name = null;
int pid ;
String sex= null;
String  cp= null;
int bs;
String exang = null;
String slope = null;
int ca;
int bp;
```

```
    int thalach;
    //String acno = null;
    //String trapass = null;
    RequestDispatcher rd=null;
    int i;
    public void init(ServletConfig sc )throws
ServletException
    {
    super.init(sc);
    }


    public void service (HttpServletRequest
req,HttpServletResponse res)
    throws ServletException, IOException
    {
    doPost (req, res);
    }
    public void doGet(HttpServletRequest
req,HttpServletResponse res)
    throws ServletException,IOException
    {
    doPost (req, res);
    }
    Public void doPost(HttpServletRequest
req,HttpServletResponse res)throws
ServletException,IOException
    {

    PrintWriter out=res.getWriter();
    res.setContentType("text/html");
    name = req.getParameter("fname");
    pid = Integer.parseInt(req.getParameter("fpid"));
    sex = req.getParameter("gender");
    cp = req.getParameter("select");
    bs = Integer.parseInt(req.getParameter("fblood"));
    exang = req.getParameter("exang");
    slope = req.getParameter("select2");
    ca = Integer.parseInt(req.getParameter("fca"));
    bp= Integer.parseInt(req.getParameter("fbp"));
                    thalach                         =
Integer.parseInt(req.getParameter("fthalach"));
                    // acno = req.getParameter("acno");
                    // trapass = req.getParameter("trapass");

   try {
 Class.forName("com.mysql.jdbc.Driver");
    con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/
distributor", "root", "password");
                        //String query = "insert into
pat_info values(
            st = con.prepareStatement("insert into Fakerecord
values('"+name+"','"+pid+"','"+sex+"','"+cp+"','"+bs+"','"+ex
ang+"','"+slope+"','"+ca+"','"+bp+"','"+thalach+"')");
 i = st.executeUpdate();
System.out.println("query executed");
if(i!=0){
 rd=req.getRequestDispatcher("success.jsp");
}
else
 {
 rd=req.getRequestDispatcher("error.jsp");
}
}
catch (Exception e)
 {
rd=req.getRequestDispatcher("error.html");
e.printStackTrace();
 }
 rd.forward(req, res);
 }
}
```

VI. Conclusion and Future work

In a perfect world, there would be no need to hand over sensitive data to agents that may unknowingly or maliciously leak it. And even if we had to hand over sensitive data, in a perfect world, we could watermark each object so that we could trace its origins with absolute certainty.

Our model is relatively simple, but we believe that it captures the essential trade-offs. The algorithms we have presented implement a variety of data distribution strategies that can improve the distributor's chances of identifying a leaker. We have shown that distributing objects judiciously can make a significant difference in identifying guilty agents, especially in cases where there is large overlap in the data that agents must receive.

Our future work includes the investigation of agent guilt models that capture leakage scenarios. Another open problem is the extension of our allocation strategies so that they can handle agent requests in an online fashion.

References

[1] A.subbiah and D.M.Blough.An Approach for fault tolerant and secure data storage in collaborative Work environments.

[2].B.Mungamuru and H.Garcia-molina,"privacy,preservation and Performance: The 3 p's of Distributed Data Management," technical report, Stanford univ.,2008

[3] M. Atallah and s.Wagstaff. Watermarking with quadratic residues. In proc.of IS&T/SPIE Conference on Security and Watermarking of Multimedia Contents, January 1999.

[4] P. Buneman and W.-C. Tan, "Provenance in Databases," Proc. ACM SIGMOD, pp. 1171-1173, 2007

[5] S.Katzenbeisser and F.A.peticolas,editors. Information Hiding Techniques for Steganography and Digital Watermarking. Artech House,2000.

[6] R. Agrawal and J. Kiernan, "Watermarking Relational Databases," Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), VLDB Endowment, pp. 155-166, 2002.

[7] Y. Cui and J. Widom, "Lineage Tracing for General Data Warehouse Transformations," The VLDB J., vol. 12, pp. 41-58, 2003.