A MODEL FOR IMPROVING GUILT PROBABILITIES TO IDENTIFY DATA LEACKAGES

Gogineni.Rajesh Chandra[1], Ch.Subba Rao[2]

[1]*Asst. Professor, Dept. Of ECM, KL University,Guntur, A.P, India*
[2]*Associate Professor, Dept. of C.S.E, Quba College of Engineering, Venkatachalam,Nellore,A.P, India*
grajeshchandra@gmail.com

Abstract

Some Times Sensitive Data may be distributes by a data distributor to a set of trusted agents (third parties), But Some of the distributed data we may find in an unauthorized website or systems. So the distributor must assess the probability of data leakage from its trusted agents. So in this paper we propose a model for assessing the guilt probability of agents, in a way that improves the chances of identifying a data leakage. We also propose algorithms for distributing objects to agents. Finally we also consider the option of adding fake data objects to the distributed data set. Such objects do not correspond to the real entities but appear realistic to the trusted agents. These fake objects used like watermarking for the entire dataset. These schemes do not make any alternations of the released data

Index Terms: *Allocation strategies, data leakage, data privacy, fake records, leakage model.*

---------- *** ----------

1. INTRODUCTION

In business sometimes we handed our sensitive to trusted third parties, i.e. agents. Owner of the data called distributor. Our goal is to detect when the agent is going to leak the data and also detect who is the agent that going to leak the data, make the agent guilty. Traditionally leakage detection is handled by the Watermarking, e.g., a unique code is embedded in each distributed copy. If that copy is later discovered in the hands of an authorized party, the leaker can be identified. These Watermarks involve some modification of the original data. Furthermore watermarks can sometimes be destroyed if the data recipient is malicious.

We present different algorithms for distributing objects to agents, in a way that improves our chances of identifying a leaker. We also consider the option of adding fake objects to the distributed set. These fake objects act as watermark for the entire distributed set, without modifying any individual member.

2. OVERALL DESCRIPTION

2 .1 Fake Objects

The fake objects or fake tuples which look like real data objects with the distributor. These fake objects are created by the distributor. The distributor may be able to add fake objects to the distributed data in order to improve his effectiveness in detecting guilty agents.

Creation: The creation of a fake object for agent Ui as a black-box function CREATEFAKEOBJECT (Ri, Fi, Condi), that takes as input as 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 U's condition. Set Ri is needed as input so that the created fake object is not only valid but also indistinguishable from other real object.

2 .b Optimization

The distributor objective is able to detect an agent who leaks any portion of his data.We define the difference function to state formally the distributor's objective as

$$\Delta(i,j) = Pr\{G_i|R_i\} - Pr\{G_j|R_i\} \qquad i,j = 1,\ldots,n$$

Where $P_r\{G_i|R_i\}$ is probability that agent U_j is guilty if the distributor discovers a leaked table S that contains all R_i objects. Assuming that the R_i sets satisfy the agents' requests, we can express the problem as a multi-criterion optimization problem:

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{maximize}} \quad (\ldots,\Delta(i,j),\ldots) \quad i \neq j$$

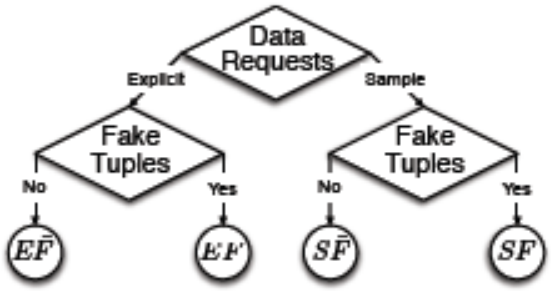

Fig 1: Leakage problem instances

EF^{l} - Explicit Request where fake tuples doesn't allow, EF – Explicit Request where fake tuples allow

SF^{l} - Sample Request where fake tuples doesn't allow, *SF* – Sample Request where fake tuples allow

Sum-Objective:

The sum objective that allow the distributor to detect the guilty agent on average (over all different agents) with higher confidence than any other distribution.

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{minimize}} \quad \sum_{i=1}^{n} \frac{1}{|R_i|} \sum_{\substack{j=1 \\ j\neq i}}^{n} |R_i \cap R_j|$$

Max-Objective:

The max objective yields the solution that guarantees that the distributor will detect
the guilty agent with a certain confidence in the worst case.

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{minimize}} \quad \max_{\substack{i,j=1,\ldots,n \\ j\neq i}} \frac{|R_i \cap R_j|}{|R_i|}$$

3. ALLOCATION STRATEGIES

In Fig 1 we show the Allocation strategies are two ways.
1. Explicit Data Requests
2. Sample Data Requests

3. a Explicit Data Requests:

In this allocation strategy, agent request distributor data objects on a constraint i.e. distributor had to distribute data objects to agent satisfying the specified condition. For e.g. Agent request distributor for patient records with constraint "patient with disease tumor".

Explicit request R_i = EXPLICIT (T, $cond_i$): Agent U_i receives all the T objects that satisfy $cond_i$.

R_i – Requested subset of objects by agent U_i in T.

T - Objects with the distributor i.e., tuples in a relation or relations in a database.

In the problems of class $EF^{\bar{l}}$ the distributor is not allowed to add fake objects to the distributor data there is nothing to optimize.

In *EF* problems, objective values are initialized by agent's data requests.

We present *Algorithm* 1 and 2 a strategy for randomly allocating fake objects.

Algorithm 1 Allocation for Explicit Data Requests (*EF*):

Input: $R_1 \ldots R_n$, $cond_1 \ldots cond_n$, $b_1 \ldots b_n$, B // B – fake objects created by distributor,
b_i – fake objects agent U_i can receive

Output: $R_1 \ldots R_n$, $F_1 \ldots F_n$ // F_i – fake object received by selected agent U_i

1: **R**← Ø // Agents that can receive fake objects
2: **for** $i = 1 \ldots n$ **do**
3: **if** $b_i > 0$ **then**
4: **R** ← **R**∪{i} // i – Agent that was selected to add fake objects
5: Fi ← Ø
6: **while** $B > 0$ **do**
7: i ← SELECTAGENT(**R**, $R_1 \ldots R_n$) // i – selected agent either by random selection or by optimal selection
8: f ← CREATEFAKEOBJEC (R_i, F_i , $cond_i$) // black box function for fake object creation
9: R_i ← R_i ∪ {f} // f – Fake object that was created for agent U_i is inserted to f
10: F_i ← F_i ∪ {f}
11: b_i ← b_i – 1

12: **if** $b_i = 0$ **then**
13: R ← R \ $\{R_i\}$
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

Algorithm 1 and 2 is a strategy for randomly allocating fake objects. Algorithm 1 is a general "driver" that will be used by other strategies, while Algorithm 2 actually performs the random selection.

In lines **1-5**, Algorithm 1 finds agents that are eligible to receiving objects in O (n) time. Then in 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).

Algorithm 2 provides optimal solution if $B \geq \Sigma^{n}_{i=1} b_i$ if $B < \Sigma^{n}_{i=1} b_i$ (As in our e.g1. where B=1 < ($b_1 + b_2$) = 2). Algorithm 2 just selects at random the agents that are provided with fake objects.

We denote the combination of *Algorithm 1 and 3* as e-optimal.

Algorithm 3 Agent Selection for e-optimal

1: **function** SELECTAGENT(R, $R_1 \ldots R_n$)
2: i ← argmax $(1/|R_{i'}| - 1/(|R_{i'}| + 1)) \Sigma_j |R_{i'} \cap R_j|$
i':$R_{i'} \in$ R
3 **return** i

In the above **e.g. 1.** If the distributor add fake object to R_2 instead of R_1 the sum-objective would be ½ + ½ = 1 < 1.33.

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 in O(n^2) in every iteration. The overall running time of e-optimal is O ($n + n^2B$) = O (n^2B).

3. b Sample Data Requests:

In this allocation strategy agent doesn't demand data objects, distributor itself distribute sample of data objects to agent. For e.g., agent request 1000 patient records, distributor will distribute patient records with any disease by picking samples randomly.

Sample request R_i = SAMPLE (T, m_i): Any subset of mi records from T objects that satisfy $cond_i$.

R_i – Requested subset of objects by agent U_i in T.

T – Objects with the distributor i.e., tuples in a relation or relations in a database.

m_i – sample of objects requested by agent U_i.

In sample data requests, each agent U_i may receive any T subset out of $\binom{|T|}{mi}$ different ones. Hence there are $\Pi_{i=1}^{n}\binom{|T|}{mi}$ different object allocations.

In sample data requests 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 there are $\Pi_{i=1}^{n}\binom{|T|}{mi}/|T|!$ different allocations.

Here we only deal with problems of class SF', but our algorithms are applicable to SF problems as well.

Algorithm 4 Allocation for Sample Data Requests (SF')

Input: $m_1 \ldots m_n$, $|T|$ // T- data set having objects with distributor
// assuming $m_i \leq |T|$ //m_i – sample data objects distributed to agent

Output: $R_1 \ldots R_n$ // R_i – Data set allocated to agent i which have m_i sample data objects
1: $a \leftarrow 0_{|T|}$ // $a[k]$: number of agents who have received object t_k
2: $R_1 \leftarrow \emptyset \ldots R_n \leftarrow \emptyset$
3: *remaining* ← $\Sigma_{i=1}^{n} m_i$ // No of sample sets that we have to distribute to agents
4: **while** *remaining* > 0 **do**
5: **for all** $i = 1 \ldots n$: $|R_i| < m_i$ **do**
6: k ← SELECTOBJECT (i, R_i) // may also use additional parameters
7: $R_i \leftarrow R_i \cup \{t_k\}$
8: $a[k] \leftarrow a[k] + 1$ // $a[k]$ – No of agents who receive object t_k
9: *remaining* ← *remaining – 1*
Algorithm 5 shows function SELECTOBJECT for s-random

Algorithm 5 Object Selection for s-random

1: **function** SELECTOBJECT (i, R_i)

2: $k \leftarrow$ select at random an element from set $\{k^l \mid t_{k^l} \notin R_i\}$
// $t_{k^l} \notin R_i$ – to avoid repetition of same element in the set of R_i
3: **return** k

We present s-random in two parts: Algorithm 4 is a general allocation algorithm that is used by other algorithms. In line **6** of Algorithm 4 there is a call to function SELECTOBJECT () whose implementation differentiates algorithms that rely on Algorithm 4.

In s-random we introduce vector $a \in N^{|T|}$ that shows object sharing distribution. Algorithm s-random allocates objects to agents in a round-robin fashion. In lines **1-2** initializations of vectors will be done. The main loop in lines **4-9** is executed while there are still data objects (*remaining*>0) to be allocated to agents.

In case of random selection we maintain in memory a set $\{k^l \mid t_{k^l} \notin R_i\}$ for each agent U_i. The running time of the algorithm is $O(T \Sigma_{i=1}^{n} m_i)$.

Algorithm 6 Object Selection for s-overlap

1: **function** SELECTOBJECT (i, R_i, a)
2: $K \leftarrow \{ k \mid k = \text{argmin } a[k^l] \}$
3: $k \leftarrow$ select at random an element from set $\{ k^l \mid k^l \in K \cap t_{k^l} \notin R_i \}$ // element that was in K and t_{k^l} was not allocated to R_i
4: **return** k

Algorithm s-random may yield a poor data allocation. For e.g., It is possible that s-random may provide all the three agents same object if the distributor set T has three objects and three agents to distribute. Such type of allocation maximizes the sum-objective and max-objective instead of minimizing.

Using Algorithm 6, in each iteration of Algorithm 4 we provide agent U_i with an object that has been given to smallest number of agents i.e. t_k object will be given less number of agents. If agents asks for fewer objects than $|T|$, Algorithm 6 will return in every iteration an object that no agent has received so far i.e., each agent receive distinct objects. The total running time of the algorithm i.e. 4 and 6 is $O(\Sigma_{i=1}^{n} m_i)$.

Algorithm 7 Object Selection for s-max

1: **function** SELECTOBJECT (i, $R_1 \ldots R_n$, $m_1 \ldots m_n$)
2: *min_overlap* ← 1 // the minimum out of the maximum relative overlaps that the allocation of different objects to U_i yields
3: **for** $k \in \{k^l \mid t_{k^l} \notin R_i\}$ **do**
4: *max_ rel _ov* ← 0 //the maximum relative overlap between R_i and any set R_j that the allocation to t_k to U_i yields
5: **for all** j=1… n: $j \neq i$ and $t_k \in R_j$ do
6: *abs_ov* ← *abs_ov*/*min* (m_i ,m_j)
7: *rel_ov* ← *abs_ov*/*min* (m_i ,m_j)
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*

Algorithm s-overlap is optimal for the max-objective optimization only if $\Sigma_{i=1}^{n} m_i \leq |T|$, also s-random ignore this objective.

For e.g., distributor set T contains four objects and there are four agents requesting two sample data objects each i.e. here $\Sigma_{i=1}^{n} m_i$ is 2*4 =8 where T is 4 i.e. 8>4. The aforementioned algorithms may produce the following data allocation:

$R_1 = \{t_1, t_2\}$, $R_2 = \{t_1, t_2\}$, $R_3 = \{t_3, t_4\}$ and $R_4 = \{t_3, t_4\}$

Although such an allocation minimizes the sum-objective, it allocates identical sets to two agent pairs, if an agent leaks his values, he will be equally guilty with an innocent agent.
To improve the worst case behavior we present a new algorithm i.e. s-max that builds upon Algorithm 4 that we used in s-random and s-overlap.

In this s-max 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 algorithm to the above example, after the first five main loop iteration in Algorithm 4 the R_i sets are:

$R_1 = \{t_1, t_2\}$, $R_2 = \{t_2\}$, $R_3 = \{t_3\}$ and $R_4 = \{t_4\}$

In the next iteration function SELECTOBJECT() must decide which object to allocate to agent U_2.We see only objects t_3 and t_4 are good candidates, since allocating t_1 to t_2 will yield a full overlap of R_1 and R_2. Function SELECTOBJECT () of s-max returns indeed t_3 or t_4.

The running time of SELECTOBJECT() is $O(|T| n)$,since the external loop in lines **3-12** in s-max iterates over all objects that agent U_i has not received and the internal loop in lines **5-8** over all agents.

The s-max is optimal for the sum-objective and the max-objective in problems where $M < |T|$. It is also optimal for max-objective if $|T| \leq M \leq 2|T|$ or $m_1=m_2=\ldots.=m_n$.

4. EXPERIMENTAL RESULTS

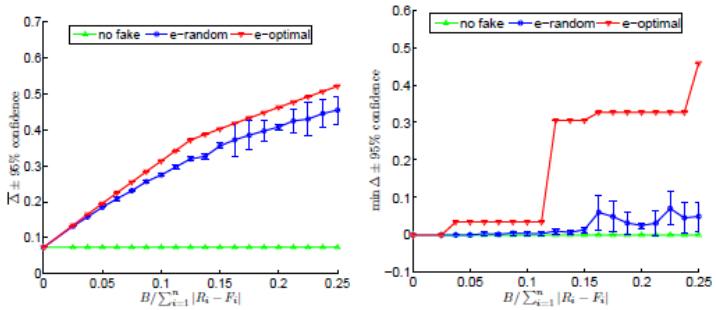

Fig2: Evaluation of Explicit Data Request Algorithms

In this section we evaluated 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. We evaluate a given allocation with the following objective scalarizations as metrics:

$$\overline{\Delta} := \frac{\sum_{\substack{i,j=1,\ldots,n \\ i \neq j}} \Delta(i,j)}{n(n-1)}$$

$$\min \Delta := \min_{\substack{i,j=1,\ldots,n \\ i \neq j}} \Delta(i,j)$$

Metric $\overline{\Delta}$ is the average of $\Delta(i, j)$ values for a given allocation and it shows how successful the guilt detection is on average for this allocation. Hence, we expect that an algorithm that is designed to minimize the sum-objective will maximize $\overline{\Delta}$. Metric min_ is the minimum $\Delta(i, j)$ value and it corresponds to the case where agent U_i has leaked his data and both U_i and another agent U_j have very similar guilt probabilities. If min Δ is small, then we will be unable to identify U_i as the leaker, versus U_j. 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.

4.b Explicit Requests

In the first place, the goal of these experiments was to see whether fake objects in the distributed data sets yield significant improvement in our chances of detecting a guilty agent. In the second place, we wanted to evaluate our e-optimal algorithm relative to a random allocation. In our scenarios we have a set of $|T|$= 10 objects for which there are requests by n = 10 different agents. We assume that each agent requests 8 particular objects out of these 10. Hence, each object is shared on average among $\frac{\sum_{i=1}^{n}|R_i|}{|T|}$ = 8 agents. Such scenarios yield very similar agent guilt probabilities and it is important to add fake objects. We generated a random scenario that yielded

4.a Metrics

We implemented the presented allocation algorithms in Python and we conducted experiments with simulated data leakage problems to evaluate their performance.

$\overline{\Delta}$ = 0.073 and min Δ = 0.35 and we applied the algorithms e-random and e-optimal to distribute fake objects to the agents. We varied the number B of distributed fake objects from 2 to 20 and for each value of B we ran both algorithms to allocate the fake objects to agents. We ran e-optimal once for each value of B, since it is a deterministic algorithm. Algorithm e-random is randomized and we ran it 10 times for each value of B. The results we present are the average over the 10 runs.

Figure 2(a) shows how fake object allocation can affect $\overline{\Delta}$. There are three curves in the plot. The solid curve is constant and shows the $\overline{\Delta}$ value for an allocation without fake objects (totally defined by agents' requests). The other two curves look at algorithms e-optimal and e-random. The y-axis shows $\overline{\Delta}$ and the x-axis show the ratio of the number of distributed fake objects to the total number of objects that the agents explicitly request.

We observe that distributing fake objects can significantly improve on average the chances of detecting a guilty agent. Even the random allocation of approximately 10% to 15% fake objects yields

$\overline{\Delta}$ > 0:3. The use of e-optimal improves $\overline{\Delta}$ further, since the e-optimal curve is consistently over the 95% confidence intervals of e-random. The performance difference between the two algorithms would be greater if the agents did not request the same number of objects, since this symmetry

allows non-smart fake object allocations to be more effective than in asymmetric scenarios. However, we do not study more this issue here, since the advantages of e-optimal become obvious when we look at our second metric.

Figure 2(b) shows the value of min Δ, as a function of the fraction of fake objects. This was expected,

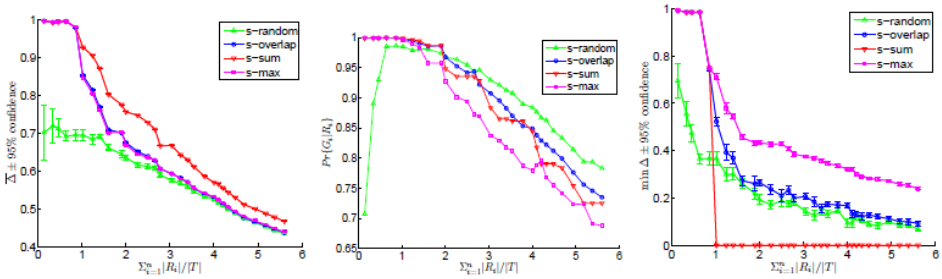

Fig 3: Evaluation of Sample Data Request Algorithms

since e-random does not take into consideration which agents "must" receive a fake object to differentiate their requests from other agents. On the contrary, algorithm e-optimal can yield min Δ > 0.3 with the allocation of approximately 10% fake objects. This improvement is very important taking into account that without fake objects values min Δ and $\overline{\Delta}$ are close to 0. That means that by allocating 10% of fake objects, the distributor can detect a guilty agent even in the worst case leakage scenario, while without fake objects he will be unsuccessful not only in the worst but also in average case.

Incidentally, the two jumps in the e-optimal curve are due to the symmetry of our scenario. Algorithm e-optimal allocates almost one fake object per agent before allocating a second fake object to one of them. The presented experiments confirmed that fake objects can have a significant impact on our chances of detecting a guilty agent. Note also that the algorithm evaluation was on the original objective. Hence, the superior performance of e-optimal (which is optimal for the approximate objective) indicates that our approximation is effective.

4 .c Sample Requests

With sample data requests agents are not interested in particular objects. The distributor is "forced" to allocate certain objects to multiple agents only if the number of requested objects $\sum_{i=1}^{n} m_i$ exceeds the number of objects in set *T*. The parameter that primarily defines the difficulty of a problem with sample data requests is $\frac{\sum_{i=1}^{n} m_i}{|T|}$.We call this ratio the *load* .

In our experimental scenarios, set *T* has 50 objects and we vary the *load*. There are two ways to vary this number: (a) assume that the number of agents is fixed and vary their sample sizes m_i, (b) vary the number of agents who request data. The latter choice captures how a real problem may evolve. The distributor may act to attract more or fewer agents for his data, but he does not have control upon agents' requests. Moreover, increasing the number of agents allows us also to increase arbitrarily the value of the *load*, while varying agents' requests poses an upper bound $n|T|$.

Our first scenario includes two agents with requests m1 and m2 that we chose uniformly at random from the interval 6. . . 15. For this scenario we ran each of the algorithms s-random (baseline), s-overlap, s-sum and s-max 10 different times, since they all include randomized steps. For each run of every algorithm we calculated $\overline{\Delta}$ and min Δ and the average over the 10 runs. The second scenario adds agent U_3 with $m_3 \sim U[6, 15]$ to the two agents of the first scenario. We repeated the 10 runs for each algorithm to allocate objects to three agents of the second scenario and calculated the two metrics values for each run. We continued adding agents and creating new scenarios to reach the number of 30 different scenarios. The last one had 31 agents. Note that we create a new scenario by adding an agent with a random request $m_i \sim U$ [6, 15] instead of assuming $m_i = 10$ for the new agent. We did that to avoid studying scenarios with equal agent sample request sizes, where certain algorithms have particular properties, e.g., s-overlap optimizes the sum-objective if requests are all the same size, but this does not hold in the general case.

In Figure 3(a) we plot the values $\overline{\Delta}$ that we found in our scenarios. There are four curves, one for each algorithm. The x-coordinate of a curve point shows the ratio of the total number of requested objects to the number of *T* objects for the scenario. The y-coordinate shows the average value of $\overline{\Delta}$ over all 10 runs. Thus, the error bar around each point shows the 95% confidence interval of $\overline{\Delta}$ values in the ten diff-erent runs. Note that algorithms s-overlap, s-sum and s-max yield $\overline{\Delta}$ values that are close to 1 if agents request in total fewer

objects than |*T*|. This was expected since in such scenarios, all three algorithms yield disjoint set allocations which is the optimal solution. In all scenarios algorithm s-sum outperforms the other ones. Algorithms s-overlap and s-max yield similar $\overline{\Delta}$ values that are between s-sum and s-random. All algorithms have $\overline{\Delta}$ around 0.5 for *load* = 4.5 which we believe is an acceptable value.

Note that in Figure 3(a), the performances of all algorithms appear to converge as the load increases. This is not true and we justify that using Figure 3(b) which shows the average guilt probability in each scenario for the actual guilty agent. Every curve point shows the mean over all 10 algorithm runs and we have omitted confidence intervals to make the plot easy to read. Note that the guilt probability for the random allocation remains significantly higher than the other algorithms for large values of the *load*. For example, if *load* ~ 5.5 algorithm s-random yields on average guilt probability 0.8 for a guilty agent and 0.8 - $\overline{\Delta}$ = 0.35 for non-guilty agent. Their relative difference is $\frac{0.8-0.35}{0.35}$ ~. 3. The corresponding probabilities that s-sum yields are 0.75 and 0.25 with relative difference $\frac{0.75-0.25}{0.25}$ = 2. Despite the fact that the absolute values of Δ converge the relative differences in the guilt probabilities between a guilty and non-guilty agents are significantly higher for s-max and s-sum compared to s-random. By comparing the curves in both figures we conclude that s-sum outperforms other algorithms for small *load* values. As the number of objects that the agents request increases its performance becomes comparable to s-max. In such cases both algorithm yield very good chances on average of detecting a guilty agents. Finally, algorithm s-overlap is inferior to them, but it still yields a significant improvement with respect to the baseline.

In Figure 3(c) we show the performance of all four algorithms with respect to *min* Δ metric. This figure is similar to Figure 3(a) and the only change is the y-axis. Algorithm s-sum now has the worst performance among all algorithms. It allocates all highly shared objects to agents who request a large sample and, consequently, these agents receive the same object sets. Two agents U_i and U_j who receive the same set have Δ (i, j) = Δ (j, i) = 0. So, if either of U_i and U_j leaks his data we cannot distinguish which of them is guilty. Random allocation has also poor performance, since as the number of agents increase, the probability that at least two agents receive many common objects becomes higher. Algorithm s-overlap limits the random allocation selection among the allocations who achieve the minimum absolute overlap summation. This fact improves on average the min Δ values, since the smaller absolute overlap reduces object sharing and, consequently, the chances that any two agents receive sets with many common objects. Algorithm s-max, which greedily allocates objects to optimize max-objective, outperforms all other algorithms and is the only that yields min Δ > 0:3 for high values of $\sum_{i=1}^{n} m_i$. Observe that the algorithm that targets at sum-objective minimization proved to be the best for the $\overline{\Delta}$ maximization and the algorithm that targets at max-objective minimization was the best for min Δ maximization.

5. CONCLUSION

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. However, in many cases we must indeed work with agents that may not be 100% trusted, and we may not be certain if a leaked object came from an agent or from some other source, since certain data cannot admit watermarks.

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.

6. FUTURE SCOPE

Our future work includes the investigation of agent guilt models that capture leakage scenarios that are not studied in this paper. For example, what is the appropriate model for cases where agents can collude and identify fake tuples?

Another open problem is the extension of our allocation strategies so that they can handle agent requests in an online fashion (the presented strategies assume that there is a fixed set of agents with requests known in advance).

REFERENCES

[1] Panagiotis Papadimitriou, Student Member, IEEE, and Hector Garcia-Molina, Member, IEEE, Data Leakage Detection, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 1, JANUARY 2011

[2] R. Agrawal and J. Kiernan. Watermarking relational databases. In VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases, pages 155–166. VLDB Endowment, 2002.

[3] P. Bonatti, S. D. C. di Vimercati, and P. Samarati. An algebra for composing access control policies. ACM Trans. Inf. Syst. Secur., 5(1):1–35, 2002.

[4] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In J. V. den Bussche and V. Vianu, editors, Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings, volume 1973 of Lecture Notes in Computer Science, pages 316–330. Springer, 2001.

[5] P. Buneman and W.-C. Tan. Provenance in databases. In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1171–1173, New York, NY, USA, 2007. ACM.

[6] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In The VLDB Journal, pages 471–480, 2001. [6] S. Czerwinski, R. Fromm, and T. Hodes. Digital music distribution and audio watermarking.

[7] F. Guo, J. Wang, Z. Zhang, X. Ye, and D. Li. Information Security Applications, pages 138–149. Springer, Berlin / Heidelberg, 2006. An Improved Algorithm to Watermark Numeric Relational Data.

[8] F. Hartung and B. Girod. Watermarking of uncompressed and compressed video. Signal Processing, 66(3):283–301, 1998.

[9] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. ACM Trans. Database Syst., 26(2):214–260, 2001.

[10] Y. Li, V. Swarup, and S. Jajodia. Fingerprinting relational databases: Schemes and specialties. IEEE Transactions on Dependable and Secure Computing, 02(1):34–45, 2005.

.