Data Leakage Detection

Panagiotis Papadimitriou, *Member, IEEE,* Hector Garcia-Molina, *Member, IEEE*

Abstract—We study the following problem: A data distributor has given sensitive data to a set of supposedly trusted agents (third parties). Some of the data is leaked and found in an unauthorized place (e.g., on the web or somebody's laptop). 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.

Index Terms—allocation strategies, data leakage, data privacy, fake records, leakage model

1 Introduction

In the course of doing business, sometimes sensitive data must be handed over to supposedly trusted third parties. For example, a hospital may give patient records to researchers who will devise new treatments. Similarly, 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. We call the owner of the data the *distributor* and the supposedly trusted third parties the *agents*. Our goal is to *detect* when the distributor's sensitive data has been *leaked* by agents, and if possible to identify the agent that leaked the data.

We consider applications where the original sensitive data cannot be perturbed. Perturbation is a very useful technique where the data is modified and made "less sensitive" before being handed to agents. For example, one can add random noise to certain attributes, or one can replace exact values by ranges [18]. However, in some cases it is important not to alter the original distributor's data. For example, if an outsourcer is doing our payroll, he must have the exact salary and customer bank account numbers. If medical researchers will be treating patients (as opposed to simply computing statistics), they may need accurate data for the patients.

Traditionally, leakage detection is handled by *watermarking*, e.g., 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. Watermarks can be very useful in some cases, but again, involve some modification of the original data. 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. (For example, the data may be found on a web site, or may be obtained through a legal discovery process.) 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 5 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 acts as a type of watermark for the entire set, without modifying any individual members. If it turns out an agent was given one or more fake objects that were leaked, then the distributor can be more confident that agent was guilty.

We start in Section 2 by introducing our problem setup and the notation we use. In the first part of the paper, Sections 4 and 5, we present a model for calculating "guilt" probabilities in cases of data leakage. Then, in the second part, Sections 6 and 7, we present strategies for data allocation to agents. Finally, in Section 8, we evaluate the strategies in different data leakage scenarios, and check whether they indeed help us to identify a leaker.

2 Problem Setup and Notation

2.1 Entities and Agents

A *distributor* owns a set $T = \{t_1, \dots, t_m\}$ of valuable data objects. The distributor wants to share some of the

- *P. Papadimitriou is with Stanford University, Stanford, CA, 94305. E-mail: papadimitriou@stanford.edu*
- *H. Garcia-Molina is with Stanford University, Stanford, CA, 94305. E-mail: hector@cs.stanford.edu*

Manuscript received October 22, 2008; revised December 14, 2009; accepted December 17, 2009.

Digital Object Indentifier 10.1109/TKDE.2010.100

objects with a set of agents $U_1, U_2, ..., U_n$, but does not wish the objects be *leaked* to other third parties. The objects in T could be of any type and size, e.g., they could be tuples in a relation, or relations in a database.

An agent U_i receives a subset of objects $R_i \subseteq T$, determined either by a sample request or an explicit request:

- Sample request $R_i = \text{SAMPLE}(T, m_i)$: Any subset of m_i records from T can be given to U_i.
- Explicit request $R_i = \text{EXPLICIT}(T, cond_i)$: Agent U_i receives *all* the T objects that satisfy $cond_i$.

Example. Say T contains customer records for a given company A. Company A hires a marketing agency U_1 to do an on-line survey of customers. Since any customers will do for the survey, U_1 requests a sample of 1000 customer records. At the same time, company A subcontracts with agent U_2 to handle billing for all California customers. Thus, U_2 receives all T records that satisfy the condition "state is California."

Although we do not discuss it here, our model can be easily extended to requests for a sample of objects that satisfy a condition (e.g., an agent wants any 100 California customer records). Also note that we do not concern ourselves with the randomness of a sample. (We assume that if a random sample is required, there are enough T records so that the to-be-presented object selection schemes can pick random records from T.)

2.2 Guilty Agents

Suppose that after giving objects to agents, the distributor discovers that a set $S \subseteq T$ has leaked. This means that some third party called the *target*, has been caught in possession of S. For example, this target may be displaying S on its web site, or perhaps as part of a legal discovery process, the target turned over S to the distributor.

Since the agents $U_1, \ldots, U_n$ have some of the data, it is reasonable to suspect them leaking the data. However, the agents can argue that they are innocent, and that the S data was obtained by the target through other means. For example, say one of the objects in S represents a customer X. Perhaps X is also a customer of some other company, and that company provided the data to the target. Or perhaps X can be reconstructed from various publicly available sources on the web.

Our goal is to estimate the likelihood that the leaked data came from the agents as opposed to other sources. Intuitively, the more data in S, the harder it is for the agents to argue they did not leak anything. Similarly, the "rarer" the objects, the harder it is to argue that the target obtained them through other means. Not only do we want to estimate the likelihood the agents leaked data, but we would also like to find out if one of them in particular was more likely to be the leaker. For instance, if one of the S objects was only given to agent U_1, while the other objects were given to all agents, we may suspect U_1 more. The model we present next captures this intuition.

We say an agent U_i is *guilty* and if it contributes one or more objects to the target. We denote the event that agent U_i is guilty as G_i and the event that agent U_i is guilty for a given leaked set S as $G_i|S$. Our next step is to estimate $Pr\{G_i|S\}$, i.e., the probability that agent U_i is guilty given evidence S.

3 Related Work

The guilt detection approach we present is related to the data provenance problem [3]: tracing the lineage of S objects implies essentially the detection of the guilty agents. Tutorial [4] provides a good overview on the research conducted in this field. Suggested solutions are domain specific, such as lineage tracing for data warehouses [5], and assume some prior knowledge on the way a data view is created out of data sources. Our problem formulation with objects and sets is more general and simplifies lineage tracing, since we do not consider any data transformation from R_i sets to S.

As far as the data allocation strategies are concerned, our work is mostly relevant to watermarking that is used as a means of establishing original ownership of distributed objects. Watermarks were initially used in images [16], video [8] and audio data [6] whose digital representation includes considerable redundancy. Recently, [1], [17], [10], [7] and other works have also studied marks insertion to relational data. Our approach and watermarking are similar in the sense of providing agents with some kind of receiver-identifying information. However, by its very nature, a watermark modifies the item being watermarked. If the object to be watermarked cannot be modified then a watermark cannot be inserted. In such cases methods that attach watermarks to the distributed data are not applicable.

Finally, there are also lots of other works on mechanisms that allow only authorized users to access sensitive data through access control policies [9], [2]. Such approaches prevent in some sense data leakage by sharing information only with trusted parties. However, these policies are restrictive and may make it impossible to satisfy agents' requests.

4 Agent Guilt Model

To compute this $Pr\{G_i|S\}$, we need an estimate for the probability that values in S can be "guessed" by the target. For instance, say some of the objects in S are emails of individuals. We can conduct an experiment and ask a person with approximately the expertise and resources of the target to find the email of say 100 individuals. If this person can find say 90 emails, then we can reasonably guess that the probability of finding one email is 0.9. On the other hand, if the objects in question are bank account numbers, the person may only discover say 20, leading to an estimate of 0.2. We call this estimate

p_t, the probability that object t can be guessed by the target.

Probability p_t is analogous to the probabilities used in designing fault-tolerant systems. That is, to estimate how likely it is that a system will be operational throughout a given period, we need the probabilities that individual components will or will not fail. A component failure in our case is the event that the target guesses an object of S. The component failure is used to compute the overall system reliability, while we use the probability of guessing to identify agents that have leaked information. The component failure probabilities are estimated based on experiments, just as we propose to estimate the p_t's. Similarly, the component probabilities are usually conservative estimates, rather than exact numbers. For example, say we use a component failure probability that is higher than the actual probability, and we design our system to provide a desired high level of reliability. Then we will know that the actual system will have at least that level of reliability, but possibly higher. In the same way, if we use p_t's that are higher than the true values, we will know that the agents will be guilty with at least the computed probabilities.

To simplify the formulas that we present in the rest of the paper, we assume that all T objects have the same p_t, which we call p. Our equations can be easily generalized to diverse p_t's though they become cumbersome to display.

Next, we make two assumptions regarding the relationship among the various leakage events. The first assumption simply states that an agent's decision to leak an object is not related to other objects. In [14] we study a scenario where the actions for different objects are related, and we study how our results are impacted by the different independence assumptions.

Assumption 1. *For all $t, t' \in S$ such that $t \neq t'$ the provenance of t is independent of the provenance of t'.*

The term provenance in this assumption statement refers to the source of a value t that appears in the leaked set. The source can be any of the agents who have t in their sets or the target itself (guessing).

To simplify our formulas, the following assumption states that joint events have a negligible probability. As we argue in the example below, this assumption gives us more conservative estimates for the guilt of agents, which is consistent with our goals.

Assumption 2. *An object $t \in S$ can only be obtained by the target in one of two ways:*

- *A single agent U_i leaked t from its own R_i set; or*
- *The target* guessed *(or obtained through other means) t without the help of any of the n agents.*

In other words, for all $t \in S$, the event that the target guesses t and the events that agent U_i ($i = 1, \ldots, n$) leaks object t are disjoint.

Before we present the general formula for computing the probability $Pr\{G_i|S\}$ that an agent U_i is guilty, we provide a simple example. Assume that the distributor set T, the agent sets R's and the target set S are:

$$T = \{t_1, t_2, t_3\},\ R_1 = \{t_1, t_2\},\ R_2 = \{t_1, t_3\},\ S = \{t_1, t_2, t_3\}.$$

In this case, all three of the distributor's objects have been leaked and appear in S. Let us first consider how the target may have obtained object t_1, which was given to both agents. From Assumption 2, the target either guessed t_1 or one of U_1 or U_2 leaked it. We know that the probability of the former event is p, so assuming that probability that each of the two agents leaked t_1 is the same we have the following cases:

- the target guessed t_1 with probability p;
- agent U_1 leaked t_1 to S with probability $(1 - p)/2$;
- agent U_2 leaked t_1 to S with probability $(1 - p)/2$

Similarly, we find that agent U_1 leaked t_2 to S with probability $1 - p$ since he is the only agent that has t_2.

Given these values, the probability that agent U_1 is not guilty, namely that U_1 did not leak either object is:

$$Pr\{\bar{G_1}|S\} = (1 - (1 - p)/2) \times (1 - (1 - p)), \tag{1}$$

and the probability that U_1 is guilty is:

$$Pr\{G_1|S\} = 1 - Pr\{\bar{G_1}\}. \tag{2}$$

Note that if Assumption 2 did not hold, our analysis would be more complex because we would need to consider joint events, e.g., the target guesses t_1 and at the same time one or two agents leak the value. In our simplified analysis we say that an agent is not guilty when the object can be guessed, regardless of whether the agent leaked the value. Since we are "not counting" instances when an agent leaks information, the simplified analysis yields conservative values (smaller probabilities).

In the general case (with our assumptions), to find the probability that an agent U_i is guilty given a set S, first we compute the probability that he leaks a single object t to S. To compute this we define the set of agents $V_t = \{U_i | t \in R_i\}$ that have t in their data sets. Then using Assumption 2 and known probability p, we have:

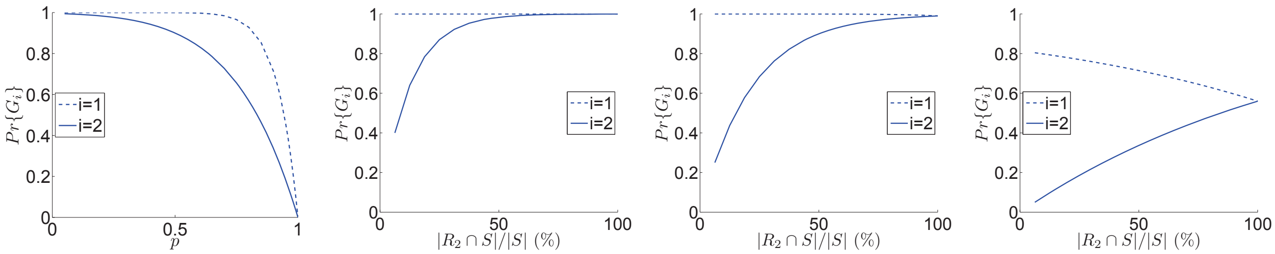
$$Pr\{\text{some agent leaked } t \text{ to } S\} = 1 - p. \tag{3}$$

Assuming that all agents that belong to V_t can leak t to S with equal probability and using Assumption 2 we obtain:

$$Pr\{U_i \text{ leaked } t \text{ to } S\} = \begin{cases} \frac{1-p}{|\mathbf{V}_t|}, & \text{if } U_i \in V_t \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Given that agent U_i is guilty if he leaks at least one value to S, with Assumption 1 and Equation 4 we can compute the probability $Pr\{G_i|S\}$ that agent U_i is guilty:

$$Pr\{G_i|S\} = 1 - \prod_{t \in S \cap R_i} \left(1 - \frac{1-p}{|V_t|}\right) \tag{5}$$



(a) $\frac{|R_2 \cap S|}{|S|} = 0.5$ (b) $p = 0.2$ (c) $p = 0.5$ (d) $p = 0.9$

Fig. 1. Guilt probability as a function of the guessing probability p (Figure (a)) and the overlap between S and R_2 (Figures (b), (c) and (d)). In all scenarios it holds that $R_1 \cap S = S$ and $|S| = 16$.

5 Guilt Model Analysis

In order to see how our model parameters interact and to check if the interactions match our intuition, in this section we study two simple scenarios. In each scenario we have a target that has obtained all the distributor's objects, i.e., $T = S$.

5.1 Impact of Probability p

In our first scenario, T contains 16 objects: all of them are given to agent U_1 and only 8 are given to a second agent U_2. We calculate the probabilities $Pr\{G_1|S\}$ and $Pr\{G_2|S\}$ for p in the range [0,1] and we present the results in Figure 1(a). The dashed line shows $Pr\{G_1|S\}$ and the solid line shows $Pr\{G_2|S\}$.

As p approaches 0, it becomes more and more unlikely that the target guessed all 16 values. Each agent has enough of the leaked data that its individual guilt approaches 1. However, as p increases in value, the probability that U_2 is guilty decreases significantly: all of U_2's 8 objects were also given to U_1, so it gets harder to blame U_2 for the leaks. On the other hand, U_2's probability of guilt remains close to 1 as p increases, since U_1 has 8 objects not seen by the other agent. At the extreme, as p approaches 1, it is very possible that the target guessed all 16 values, so the agent's probability of guilt goes to 0.

5.2 Impact of Overlap between R_i and S

In this subsection we again study two agents, one receiving all the $T = S$ data, and the second one receiving a varying fraction of the data. Figure 1(b) shows the probability of guilt for both agents, as a function of the fraction of the objects owned by U_2, i.e., as a function of $|R_2 \cap S|/|S|$. In this case, p has a low value of 0.2, and U_1 continues to have all 16 S objects. Note that in our previous scenario, U_2 has 50% of the S objects.

We see that when objects are rare ($p = 0.2$), it does not take many leaked objects before we can say U_2 is guilty with high confidence. This result matches our intuition: an agent that owns even a small number of incriminating objects is clearly suspicious.

Figures 1(c) and 1(d) show the same scenario, except for values of p equal to 0.5 and 0.9. We see clearly that the rate of increase of the guilt probability decreases as p increases. This observation again matches our intuition: As the objects become easier to guess, it takes more and more evidence of leakage (more leaked objects owned by U_2) before we can have high confidence that U_2 is guilty.

In [14] we study an additional scenario that shows how the sharing of S objects by agents affects the probabilities that they are guilty. The scenario conclusion matches our intuition: with more agents holding the replicated leaked data, it is harder to lay the blame on any one agent.

6 Data Allocation Problem

The main focus of our paper is the data allocation problem: how can the distributor "intelligently" give data to agents in order to improve the chances of detecting a guilty agent? As illustrated in Figure 2, there are four instances of this problem we address, depending on the type of data requests made by agents and whether "fake objects" are allowed.

The two types of requests we handle were defined in Section 2: sample and explicit. Fake objects are objects generated by the distributor that are *not* in set T. The objects are designed to look like real objects, and are distributed to agents together with the T objects, in order to increase the chances of detecting agents that leak data. We discuss fake objects in more detail in Section 6.1 below.

As shown in Figure 2, we represent our four problem instances with the names $E\overline{F}$, EF, $S\overline{F}$ and SF, where E stands for explicit requests, S for sample requests, F for the use of fake objects, and $\overline{F}$ for the case where fake objects are not allowed.

Note that for simplicity we are assuming that in the E problem instances, *all* agents make explicit requests,

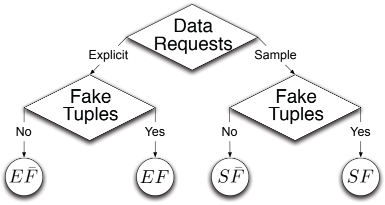


Fig. 2. Leakage Problem Instances

while in the S instances, *all* agents make sample requests. Our results can be extended to handle mixed cases, with some explicit and some sample requests. We provide here a small example to illustrate how mixed requests can be handled, but then do not elaborate further. Assume that we have two agents with requests $R_1 = \textsc{Explicit}(T, cond_1)$ and $R_2 = \textsc{Sample}(T', 1)$ where $T' = \textsc{Explicit}(T, cond_2)$. Further, say $cond_1$ is "state=CA" (objects have a state field). If agent U_2 has the same condition $cond_2 = cond_1$, we can create an equivalent problem with sample data requests on set T'. That is, our problem will be how to distribute the CA objects to two agents, with $R_1 = \textsc{Sample}(T', |T'|)$ and $R_2 = \textsc{Sample}(T', 1)$. If instead U_2 uses condition "state=NY," we can solve two different problems for sets T' and $T - T'$. In each problem we will have only one agent. Finally, if the conditions partially overlap, $R_1 \cap T' \neq \emptyset$, but $R_1 \neq T'$ we can solve three different problems for sets $R_1 - T'$, $R_1 \cap T'$ and $T' - R_1$.

6.1 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 a watermark to an image. In our case, we are perturbing the *set* of distributor objects by adding fake elements. In some applications, fake objects may cause fewer problems that perturbing real objects. For example, say the distributed data objects are medical records and the agents are hospitals. In this case, even small modifications to the records of actual patients may be undesirable. However, the addition of some fake medical records may be acceptable, since no patient matches these records, and hence no one will ever be treated based on fake records.

Our use of fake objects is inspired by the use of "trace" records in mailing lists. In this case, company A sells to company B a mailing list to be used once (e.g., to send advertisements). Company A adds trace records that contain addresses owned by company A. Thus, each time company B uses the purchased mailing list, A receives copies of the mailing. These records are a type of fake objects that help identify improper use of data.

The distributor creates and adds fake objects to the data that he distributes to agents. We let $F_i \subseteq R_i$ be the subset of fake objects that agent U_i receives. As discussed below, fake objects must be created carefully so that agents cannot distinguish them from real objects.

In many cases, the distributor may be limited in how many fake objects he can create. For example, objects may contain email addresses, and each fake email address may require the creation of an actual inbox (otherwise the agent may discover the object is fake). The inboxes can actually be monitored by the distributor: if email is received from someone other than the agent who was given the address, it is evidence that the address was leaked. Since creating and monitoring email accounts consumes resources, the distributor may have a limit of fake objects. If there is a limit, we denote it by B fake objects.

Similarly, the distributor may want to limit the number of fake objects received by each agent, so as to not arouse suspicions and to not adversely impact the agents activities. Thus, we say that the distributor can send up to b_i fake objects to agent U_i.

Creation. The creation of fake but real-looking objects is a non-trivial problem whose thorough investigation is beyond the scope of this paper. Here, we model the creation of a fake object for agent U_i as a black-box function $\textsc{CreateFakeObject}(R_i, F_i, cond_i)$ that takes as input the set of all objects R_i, the subset of fake objects F_i that U_i has received so far and $cond_i$, and returns a new fake object. This function needs $cond_i$ to produce a valid object that satisfies U_i's condition. Set R_i 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 $\textsc{CreateFakeObject}()$ has to be aware of the fake objects F_i added so far, again to ensure proper statistics.

The distributor can also use function $\textsc{CreateFakeObject}()$ 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 R_i and F_i tables respectively, and the intersection of the conditions $cond_i$'s.

Although we do not deal with the implementation of $\textsc{CreateFakeObject}()$ 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.

6.2 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 as in [13] 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 (Mungamuru et al. [11] discuss that to attain "perfect" privacy and security we have to sacrifice utility). 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 $Pr\{G_j|S = R_i\}$ or simply $Pr\{G_j|R_i\}$ is the probability that agent U_j is guilty if the distributor discovers a leaked table S that contains all R_i objects. We define the difference functions $\Delta(i,j)$ as:

$$\Delta(i,j) = Pr\{G_i|R_i\} - Pr\{G_j|R_i\} \qquad i,j = 1,\ldots,n \tag{6}$$

Note that differences Δ have non-negative values: given that set R_i contains all the leaked objects, agent U_i is at least as likely to be guilty as any other agent. Difference $\Delta(i,j)$ is positive for any agent U_j, whose set R_j does not contain all data of S. It is zero, if $R_i \subseteq R_j$. In this case the distributor will consider both agents U_i and U_j equally guilty since they have both received all the leaked objects. The larger a $\Delta(i,j)$ value is, the easier it is to identify U_i as the leaking agent. Thus, we want to distribute data so that Δ values are large:

Problem Definition. *Let the distributor have data requests from n agents. The distributor wants to give tables $R_1,\ldots,R_n$ to agents $U_1,\ldots,U_n$ respectively, so that:*

- *he satisfies agents' requests; and*
- *he maximizes the guilt probability differences $\Delta(i,j)$ for all $i,j = 1,\ldots,n$ and $i \neq j$.*

Assuming that the R_i sets satisfy the agents' requests, we can express the problem as a multi-criterion optimization problem:

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{maximize}} \quad (\ldots,\Delta(i,j),\ldots) \quad i \neq j \tag{7}$$

If the optimization problem has an optimal solution, that means that there exists an allocation $\mathcal{D}^* = \{R_1^*,\ldots,R_n^*\}$ such that any other feasible allocation $\mathcal{D} = \{R_1,\ldots,R_n\}$ yields $\Delta(i,j) \geq \Delta^*(i,j)$ for all i,j. That means that allocation $\mathcal{D}^*$ allows the distributor to discern any guilty agent with higher confidence than any other allocation, since it maximizes the probability $Pr\{G_i|R_i\}$ with respect to any other probability $Pr\{G_i|R_j\}$ with $j \neq i$.

Even if there is no optimal allocation $\mathcal{D}^*$, a multi-criterion problem has Pareto optimal allocations. If $\mathcal{D}^{po} = \{R_1^{po},\ldots,R_n^{po}\}$ is a Pareto optimal allocation, that means that there is no other allocation that yields $\Delta(i,j) \geq \Delta^{po}(i,j)$ for all i,j. In other words, if an allocation yields $\Delta(i,j) \geq \Delta^{po}(i,j)$ for some i,j, then there is some i',j' such that $\Delta(i',j') \leq \Delta^{po}(i',j')$. The choice among all the Pareto optimal allocations implicitly selects the agent(s) we want to identify.

6.3 Objective Approximation

We can approximate the objective of Equation 7 with Equation 8 that does not depend on agents' guilt probabilities and therefore on p.

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{minimize}} \quad \left(\ldots,\frac{|R_i\cap R_j|}{|R_i|},\ldots\right) \quad i \neq j \tag{8}$$

This approximation is valid if minimizing the relative overlap $\frac{|R_i\cap R_j|}{|R_i|}$ maximizes $\Delta(i,j)$. The intuitive argument for this approximation is that the fewer leaked objects set R_j contains, the less guilty agent U_j will appear compared to U_i (since $S = R_i$). The example of Section 5.2 supports our approximation. In Figure 1 we see that if $S = R_1$ the difference $Pr\{G_1|S\} - Pr\{G_2|S\}$ decreases as the relative overlap $\frac{|R_2\cap S|}{|S|}$ increases. Theorem 1 shows that a solution to problem 7 yields the solution to problem 8 if each T object is allocated to the same number of agents, regardless of who these agents are. The proof of the theorem is in [14].

Theorem 1. *If a distribution $\mathcal{D} = \{R_1,\ldots,R_n\}$ that satisfies agents' requests minimizes $\frac{|R_i\cap R_j|}{|R_i|}$ and $|V_t| = |V_{t'}|$ for all $t,t' \in T$, then $\mathcal{D}$ maximizes $\Delta(i,j)$.*

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 U_i 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. We present two different scalar versions of our problem in Equations 9a and 9b. In the rest of the paper we will refer to objective 9a as the *sum-objective* and to objective 9b as the *max-objective*.

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{minimize}} \quad \sum_{i=1}^{n}\frac{1}{|R_i|}\sum_{\substack{j=1\\ j\neq i}}^{n}|R_i\cap R_j| \tag{9a}$$

$$\underset{(\text{over } R_1,\ldots,R_n)}{\text{minimize}} \quad \max_{\substack{i,j=1,\ldots,n\\ j\neq i}}\frac{|R_i\cap R_j|}{|R_i|} \tag{9b}$$

Both scalar optimizations problems yield the optimal solution of the problem of Equation 8, if such solution exists. If there is no global optimal solution, the sum-objective yields the Pareto optimal solution that allow 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 a certain confidence in the worst case. Such guarantee may adversely impact the average performance of the distribution.

7 Allocation Strategies

In this section we describe allocation strategies that solve exactly or approximately the scalar versions of Equation 8 for the different instances presented in Figure 2. We resort to approximate solutions in cases where it is inefficient to solve accurately the optimization problem.

In Section 7.1 we deal with problems with explicit data requests and in Section 7.2 with problems with sample data requests.

The proofs of theorems that are stated in the following sections are available in [14].

7.1 Explicit Data Requests

In problems of class $E\overline{F}$ 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 $T = \{t_1, t_2\}$ and there are two agents with explicit data requests such that $R_1 = \{t_1, t_2\}$ and $R_2 = \{t_1\}$. The value of the sum-objective is in this case:

$$\sum_{i=1}^{2} \frac{1}{|R_i|} \sum_{\substack{j=1 \\ j\neq i}}^{2} |R_i \cap R_j| = \frac{1}{2} + \frac{1}{1} = 1.5.$$

The distributor cannot remove or alter the R_1 or R_2 data to decrease the overlap $R_1 \cap R_2$. However, say the distributor can create one fake object ($B = 1$) and both agents can receive one fake object ($b_1 = b_2 = 1$). In this case, the distributor can add one fake object to either R_1 or R_2 to increase the corresponding denominator of the summation term. Assume that the distributor creates a fake object f and he gives it to agent R_1. Agent U_1 has now $R_1 = \{t_1, t_2, f\}$ and $F_1 = \{f\}$ and the value of the sum-objective decreases to $\frac{1}{3} + \frac{1}{1} = 1.33 < 1.5$.

If the distributor is able to create more fake objects, he could further improve the objective. We present in Algorithm 1 and 2 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. We denote the combination of Algorithm 1 with 2 as e-random. We use e-random as our baseline in our comparisons with other algorithms for explicit data requests.

Algorithm 1 Allocation for Explicit Data Requests (EF)

```
Input: $R_1, \ldots, R_n$, $cond_1, \ldots, cond_n$, $b_1, \ldots, b_n$, $B$
Output: $R_1, \ldots, R_n$, $F_1, \ldots, F_n$
 1: $\mathbf{R} \leftarrow \emptyset$                ▷ Agents that can receive fake objects
 2: for $i = 1, \ldots, n$ do
 3:     if $b_i > 0$ then
 4:         $\mathbf{R} \leftarrow \mathbf{R} \cup \{i\}$
 5:     $F_i \leftarrow \emptyset$
 6: while $\mathrm{B} > 0$ do
 7:     $i \leftarrow$ SelectAgent($\mathbf{R}, R_1, \ldots, R_n$)
 8:     $f \leftarrow$ CreateFakeObject($R_i, F_i, cond_i$)
 9:     $R_i \leftarrow R_i \cup \{f\}$
10:     $F_i \leftarrow F_i \cup \{f\}$
11:     $b_i \leftarrow b_i - 1$
12:     if $b_i = 0$ then
13:         $\mathbf{R} \leftarrow \mathbf{R} \backslash \{R_i\}$
14:     $B \leftarrow B - 1$
```

Algorithm 2 Agent Selection for e-random

```
function SelectAgent($\mathbf{R}, R_1, \ldots, R_n$)
    $i \leftarrow$ select at random an agent from $\mathbf{R}$
    return $i$
```

Algorithm 3 Agent selection for e-optimal

```
function SelectAgent($\mathbf{R}, R_1, \ldots, R_n$)
    $i \leftarrow \underset{i' : R_{i'} \in \mathbf{R}}{\mathrm{argmax}} \left( \frac{1}{|R_{i'}|} - \frac{1}{|R_{i'}|+1} \right) \sum_j |R_{i'} \cap R_j|$
    return $i$
```

In lines 1-5, Algorithm 1 finds agents that are eligible to receiving fake objects in $\mathrm{O}(n)$ 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 $\mathrm{O}(B)$ time. Hence, the running time of the algorithm is $\mathrm{O}(n + B)$.

If $B \geq \sum_{i=1}^{n} b_i$, the algorithm minimizes every term of the objective summation by adding the maximum number b_i of fake objects to every set R_i, yielding the optimal solution. Otherwise, if $B < \sum_{i=1}^{n} b_i$ (as in our example where $B = 1 < b_1 + b_2 = 2$) 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 R_2 instead of R_1. In this case the sum-objective would be $\frac{1}{2} + \frac{1}{2} = 1 < 1.33$.

The reason why we got a greater improvement is that the addition of a fake object to R_2 has greater impact on the corresponding summation terms, since

$$\frac{1}{|R_1|} - \frac{1}{|R_1| + 1} = \frac{1}{6} < \frac{1}{|R_2|} - \frac{1}{|R_2| + 1} = \frac{1}{2}.$$

The left-hand side of the inequality corresponds to the objective improvement after the addition of a fake object to R_1 and the right-hand side to R_2. We can use this observation to improve Algorithm 1 with the use of procedure SelectAgent() of Algorithm 3. We denote the combination of Algorithms 1 and 3 as e-optimal.

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 $\mathrm{O}(n^2)$ in every iteration. The overall running time of e-optimal is $\mathrm{O}(n + n^2 B) = \mathrm{O}(n^2 B)$. Theorem 2 shows that this greedy approach finds an optimal distribution with respect to both optimization objectives defined in Equation 9.

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

7.2 Sample Data Requests

With sample data requests, each agent U_i may receive any T subset out of $\binom{|T|}{m_i}$ different ones. Hence, there are

Algorithm 4 Allocation for Sample Data Requests ($S\overline{F}$)

```
Input: $m_1, \ldots, m_n$, $|T|$                    ▷ Assuming $m_i \leq |T|$
Output: $R_1, \ldots, R_n$
1: $a \leftarrow \mathbf{0}_{|T|}$        ▷ $a[k]$:number of agents who have received object $t_k$
2: $R_1 \leftarrow \emptyset, \ldots, R_n \leftarrow \emptyset$
3: $remaining \leftarrow \sum_{i=1}^{n} m_i$
4: while $remaining > 0$ do
5:     for all $i = 1, \ldots, n : |R_i| < m_i$ do
6:         $k \leftarrow$ SELECTOBJECT($i, R_i$)          ▷ May also use additional parameters
7:         $R_i \leftarrow R_i \cup \{t_k\}$
8:         $a[k] \leftarrow a[k] + 1$
9:         $remaining \leftarrow remaining - 1$
```

Algorithm 5 Object Selection for s-random

```
function SELECTOBJECT($i, R_i$)
    $k \leftarrow$ select at random an element from set $\{k' \mid t_{k'} \notin R_i\}$
    return $k$
```

Algorithm 6 Object Selection for s-overlap

```
function SELECTOBJECT($i, R_i, a$)
    $K \leftarrow \{k \mid k = \underset{k'}{\operatorname{argmin}}\ a[k']\}$
    $k \leftarrow$ select at random an element from set $\{k' \mid k' \in K \wedge t_{k'} \notin R_i\}$
    return $k$
```

$\prod_{i=1}^{n} \binom{|T|}{m_i}$ 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 there are $\frac{\prod_{i=1}^{n} \binom{|T|}{m_i}}{|T|!}$ different allocations. The distributor's problem is to pick one out so that he optimizes his objective. In [14] we formulate the problem as a non-convex QIP that is NP-hard to solve [15].

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 subsection we will only deal with problems of class $S\overline{F}$, but our algorithms are applicable to SF problems as well.

7.2.1 Random

An object allocation that satisfies requests and ignores the distributor's objective is to give each agent U_i a randomly selected subset of T of size m_i. We denote this algorithm as 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.

In s-random we introduce vector $a \in \mathbb{N}^{|T|}$ that shows the *object sharing distribution*. In particular, element $a[k]$ shows the number of agents who receive object t_k.

Algorithm s-random allocates objects to agents in a round-robin fashion. After the initialization of vectors d and a in lines 1-2 of Algorithm 4, the main loop in lines 4-9 is executed while there are still data objects ($remaining > 0$) to be allocated to agents. In each iteration of this loop (lines 5-9) the algorithm uses function SELECTOBJECT() to find a random object to allocate to agent U_i. This loop iterates over all agents who have not received the number of data objects they have requested.

The running time of the algorithm is $O(\tau \sum_{i=1}^{n} m_i)$ and depends on the running time τ of the object selection function SELECTOBJECT(). In case of random selection we can have $\tau = O(1)$ by keeping in memory a set $\{k' \mid t_{k'} \notin R_i\}$ for each agent U_i.

Algorithm s-random may yield a poor data allocation. Say, for example that the distributor set T has three objects and there are three agents who request one object each. It is possible that s-random provides all three agents with the same object. Such an allocation maximizes both objectives 9a and 9b instead of minimizing them.

7.2.2 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 as s-overlap. Using Algorithm 6, in each iteration of Algorithm 4 we provide agent U_i 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.

The running time of Algorithm 6 is $O(1)$ if we keep in memory the set $\{k | k = \underset{k'}{\operatorname{argmin}}\ a[k']\}$. This set contains initially all indices $\{1, \ldots, |T|\}$, since $a[k] = 0$ for all $k = 1, \ldots, |T|$. 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 $\{1, \ldots, |T|\}$, since at this point $a[k] = 1$ for all $k = 1, \ldots, |T|$. The total running time of algorithm s-random is thus $O(\sum_{i=1}^{n} m_i)$.

Let $M = \sum_{i=1}^{n} m_i$. If $M \leq |T|$, 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 yields an object sharing distribution such that:

$$a[k] = \begin{cases} M \div |T| + 1 & \text{for } M \bmod |T| \text{ entries of vector } a \\ M \div |T| & \text{for the rest} \end{cases}$$

Theorem 3. *In general, Algorithm s-overlap does not minimize sum-objective. However, s-overlap does minimize the sum of overlaps, i.e., $\sum_{\substack{i,j=1 \\ j \neq i}}^{n} |R_i \cap R_j|$. If requests are all of the same size ($m_1 = \cdots = m_n$) then s-overlap minimizes the sum-objective.*

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 $m_1 = m_2 = 2$ and $m_3 = m_4 = 1$. A possible data allocation from s-overlap is

$$R_1 = \{t_1, t_2\}, R_2 = \{t_3, t_4\}, R_3 = \{t_1\}, R_4 = \{t_3\}. \quad (10)$$

Allocation 10 yields:

$$\sum_{i=1}^{4} \frac{1}{|R_i|} \sum_{\substack{j=1 \\ j \neq i}}^{4} |R_i \cap R_j| = \frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} = 3.$$

With this allocation, we see that if agent U_3 leaks his data we will equally suspect agents U_1 and U_3. Moreover, if agent U_1 leaks his data we will suspect U_3 with high probability, since he has half of the leaked data. The situation is similar for agents U_2 and U_4.

However, the following object allocation

$$R_1 = \{t_1, t_2\}, R_2 = \{t_1, t_2\}, R_3 = \{t_3\}, R_4 = \{t_4\} \quad (11)$$

yields a sum-objective equal to $\frac{2}{2} + \frac{2}{2} + 0 + 0 = 2 < 3$, which shows that the first allocation is not optimal. With this allocation, we will equally suspect agents U_1 and U_2 if either of them leaks his data. However, if either U_3 or U_4 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.

7.2.3 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 small 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 (See Section 5.2). In [14] we provide an algorithm that implements this intuition and we denote it s-sum. Although we evaluate this algorithm in Section 8, we do not present the pseudocode here due to space limitations.

7.2.4 Approximate Max-Objective Minimization

Algorithm s-overlap is optimal for the max-objective optimization only if $\sum_i^n m_i \leq |T|$. Note also that s-sum as well as s-random ignore this objective. Say, for example, that set T contains for objects and there are 4 agents, each requesting a sample of 2 data objects. The aforementioned algorithms may produce the following data allocation:

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

Although such an allocation minimizes the sum-objective, it allocates identical sets to two agent pairs. Consequently, if an agent leaks his values, he will be equally guilty with an innocent agent.

Algorithm 7 Object Selection for s-max

```
 1: function SelectObject($i, R_1, \ldots, R_n, m_1, \ldots, m_n$)
 2:     $min\_overlap \leftarrow 1$        ▷ the minimum out of the maximum relative overlaps that the allocations of different objects to $U_i$ yield
 3:     for $k \in \{k' \mid t_{k'} \notin R_i\}$ do
 4:         $max\_rel\_ov \leftarrow 0$ ▷ the maximum relative overlap between $R_i$ and any set $R_j$ that the allocation of $t_k$ to $U_i$ yields
 5:         for all $j = 1, \ldots, n : j \neq i$ and $t_k \in R_j$ do
 6:             $abs\_ov \leftarrow |R_i \cap R_j| + 1$
 7:             $rel\_ov \leftarrow abs\_ov / min(m_i, m_j)$
 8:             $max\_rel\_ov \leftarrow$ Max($max\_rel\_ov, rel\_ov$)
 9:         if $max\_rel\_ov \leq min\_overlap$ then
10:             $min\_overlap \leftarrow max\_rel\_ov$
11:             $ret\_k \leftarrow k$
12:     return $ret\_k$
```

To improve the worst-case behavior we present a new algorithm that builds upon Algorithm 4 that we used in s-random and s-overlap. We define a new SelectObject() procedure in Algorithm 7. We denote the new algorithm 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 R_i sets are:

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

In the next iteration function SelectObject() must decide which object to allocate to agent U_2. We see that only objects t_3 and t_4 are good candidates, since allocating t_1 to U_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 iterates over all objects that agent U_i has not received and the internal loop in lines 5-8 over all agents. This running time calculation implies that we keep the overlap sizes $R_i \cap R_j$ for all agents in a two-dimensional array that we update after every object allocation.

It can be shown that algorithm s-max is optimal for the sum-objective and the max-objective in problems where $M \leq |T|$. It is also optimal for the max-objective if $|T| \leq M \leq 2|T|$ or $m_1 = m_2 = \cdots = m_n$.

8 Experimental Results

We implemented the presented allocation algorithms in Python and we conducted experiments with simulated data leakage problems to evaluate their performance. In Section 8.1 we present the metrics we use for the algorithm evaluation and in Sections 8.2 and 8.3 we present the evaluation for sample requests and explicit data requests respectively.

8.1 Metrics

In Section 7 we presented algorithms to optimize the problem of Equation 8 that is an approximation to the original optimization problem of Equation 7. 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 Equation 7 are the Δ difference functions. Note that there are $n(n-1)$ objectives, since for each agent U_i there are $n-1$ differences $\Delta(i,j)$ for $j=1,\ldots,n$ and $j \neq i$. We evaluate a given allocation with the following objective scalarizations as metrics:

$$\overline{\Delta} := \frac{\sum_{\substack{i,j=1,\ldots,n\\ i\neq j}} \Delta(i,j)}{n(n-1)} \tag{12a}$$

$$\min\Delta := \min_{\substack{i,j=1,\ldots,n\\ i\neq j}} \Delta(i,j) \tag{12b}$$

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. For example, if $\overline{\Delta}=0.4$, then on average the probability $Pr\{G_i|R_i\}$ for the actual guilty agent will be 0.4 higher than the probabilities of non-guilty agents. Note that this scalar version of the original problem objective is analogous to the sum-objective scalarization of the problem of Equation 8. Hence, we expect that an algorithm that is designed to minimize the sum-objective will maximize $\overline{\Delta}$.

Metric $\min\Delta$ 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 $\min\Delta$ is small, then we will be unable to identify U_i as the leaker, versus U_j. If $\min\Delta$ 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 non-guilty 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 $Pr\{G_i|R_i\} = 0.9$ is enough to arouse our suspicion that agent U_i leaked data. Furthermore, say the difference between $Pr\{G_i|R_i\}$ and any other $Pr\{G_j|R_i\}$ is at least 0.3. In other words, the guilty agent is $(0.9-0.6)/0.6\times100\% = 50\%$ more likely to be guilty compared to the other agents. In this case we may be willing to take action against U_i (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 Δ values.

To calculate the guilt probabilities and Δ 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 need to find an agent guilty.

8.2 Explicit Requests

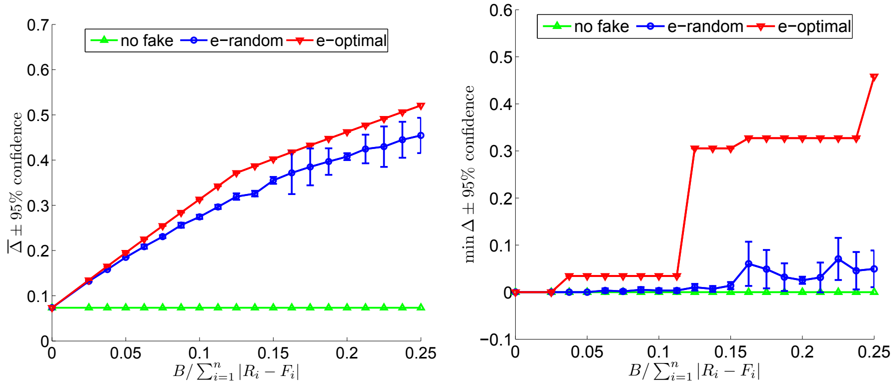
In the first place, the goal of these experiments was to see whether fake objects in the distributed data sets yield significant improvement in our chances of detecting a guilty agent. In the second place, we wanted to evaluate our e-optimal algorithm relative to a random allocation.

We focus on scenarios with a few objects that are shared among multiple agents. These are the most interesting scenarios, since object sharing makes it difficult to distinguish a guilty from non-guilty agents. Scenarios with more objects to distribute or scenarios with objects shared among fewer agents are obviously easier to handle. As far as scenarios with many objects to distribute and many overlapping agent requests are concerned, they are similar to the scenarios we study, since we can map them to the distribution of many small subsets.

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 $\overline{\Delta}=0.073$ and $\min\Delta=0.35$ and we applied the algorithms e-random and e-optimal to distribute fake objects to the agents (See [14] for other randomly generated scenarios with the same parameters). 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 3(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 shows 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



(a) Average $\overline{\Delta}$

(b) Average $\min\Delta$

Fig. 3. Evaluation of Explicit Data Request Algorithms

more this issue here, since the advantages of e-optimal become obvious when we look at our second metric.

Figure 3(b) shows the value of $\min\Delta$, as a function of the fraction of fake objects. The plot shows that random allocation will yield an insignificant improvement in our chances of detecting a guilty agent in the worst case scenario. This was expected, 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 $\min\Delta > 0.3$ with the allocation of approximately 10% fake objects. This improvement is very important taking into account that without fake objects values $\min\Delta$ 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.

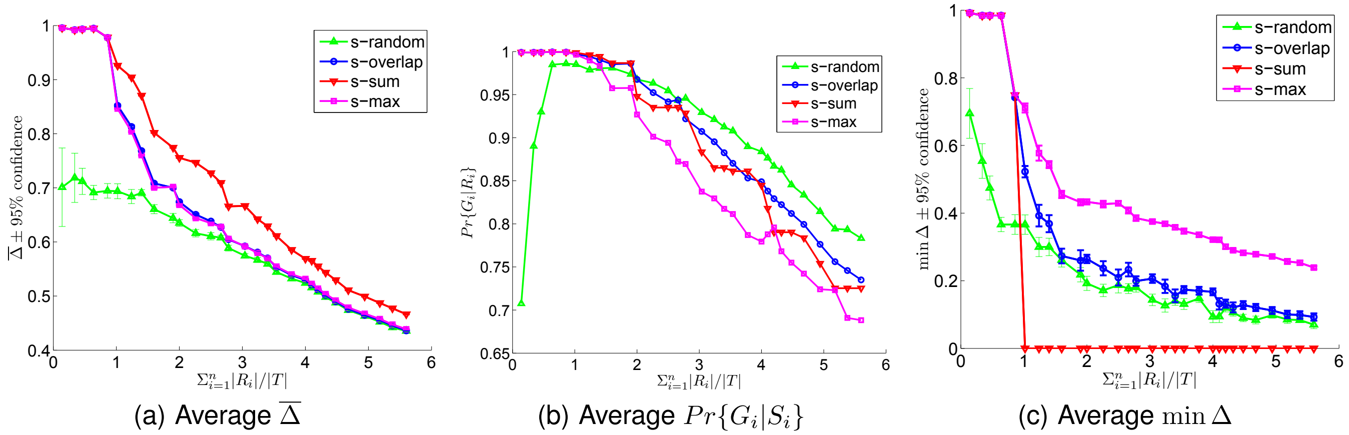
8.3 Sample Requests

With sample data requests agents are not interested in particular objects. Hence, object sharing is not explicitly defined by their requests. 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 more data objects the agents request in total, the more recipients on average an object has; and the more objects are shared among different agents, the more difficult it is to detect a guilty agent. Consequently, the parameter that primarily defines the difficulty of a problem with sample data requests is the ratio $\frac{\sum_{i=1}^{n} m_i}{|T|}$. We call this ratio the *load*. Note also, that the absolute values of $m_1, \ldots, m_n$ and $|T|$ play a less important role than the relative values $m_i/|T|$. Say, for example, that $T = 99$ and algorithm X yields a good allocation for the agents' requests $m_1 = 66$ and $m_2 = m_3 = 33$. Note that for any $|T|$ and $m_1/|T| = 2/3$, $m_2/|T| = m_3/|T| = 1/3$ the problem is essentially similar and algorithm X would still yield a good allocation.

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 m_1 and m_2 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 $min\Delta$ 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 4(a) we plot the values $\overline{\Delta}$ that we found in our scenarios. There are four curves, one for each



(a) Average $\overline{\Delta}$

(b) Average $Pr\{G_i|S_i\}$

(c) Average $\min\Delta$

Fig. 4. Evaluation of Sample Data Request Algorithms

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 different 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 4(a), the performances of all algorithms appear to converge as the $load$ increases. This is not true and we justify that using Figure 4(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 $load \approx 5.5$ 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 $\frac{0.8-0.35}{0.35} \approx 1.3$. 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 $\overline{\Delta}$ 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 4(c) we show the performance of all four algorithms with respect to $min\Delta$ metric. This figure is is similar to Figure 4(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 $\Delta(i,j) = \Delta(j,i) = 0$. 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 $\min\Delta$ 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 $\min\Delta > 0.3$ 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 $min\Delta$ maximization. These facts confirm that the approximation of objective 7 with 8 is effective.

9 Conclusions

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.

In spite of these difficulties, we have shown it is possible to assess the likelihood that an agent is responsible for a leak, based on the overlap of his data with the leaked data and the data of other agents, and based on the probability that objects can be "guessed" by other means. Our model is relatively simple, but we believe 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 that are not studied in this paper. For example, what is the appropriate model for cases where agents can collude and identify fake tuples? A preliminary discussion of such a model is available in [14]. 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).

Acknowledgments

This work was supported by NSF Grant CFF-0424422 and an Onassis Foundation Scholarship. We would like thank Paul Heymann for his help with running the non-polynomial guilt model detection algorithm that we present in the Appendix of [14] on a Hadoop cluster. We also thank Ioannis Antonellis for fruitful discussions and his comments on earlier versions of this paper.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[11] B. Mungamuru and H. Garcia-Molina. Privacy, preservation and performance: The 3 p's of distributed data management. Technical report, Stanford University, 2008.

[12] V. N. Murty. Counting the integer solutions of a linear equation with unit coefficients. *Mathematics Magazine*, 54(2):79–81, 1981.

[13] S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 151–162. VLDB Endowment, 2006.

[14] P. Papadimitriou and H. Garcia-Molina. Data leakage detection. Technical report, Stanford University, 2008.

[15] P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[16] J. J. K. O. Ruanaidh, W. J. Dowling, and F. M. Boland. Watermarking digital images for copyright protection. *I.E.E. Proceedings on Vision, Signal and Image Processing*, 143(4):250–256, 1996.

[17] R. Sion, M. Atallah, and S. Prabhakar. Rights protection for relational data. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 98–109, New York, NY, USA, 2003. ACM.

[18] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression, 2002.

Panagiotis Papadimitriou is a PhD student in the Department of Electrical Engineering at Stanford University, Stanford, California. He received a Diploma in electrical and computer engineering from National Techical University of Athens in 2006 and a MS in electrical engineering from Stanford University in 2008. His research interests include internet advertising, data mining, data privacy and web search.

Hector Garcia-Molina is the Leonard Bosack and Sandra Lerner Professor in the Departments of Computer Science and Electrical Engineering at Stanford University, Stanford, California. He was the chairman of the Computer Science Department from January 2001 to December 2004. From 1997 to 2001 he was a member the President's Information Technology Advisory Committee (PITAC). From August 1994 to December 1997 he was the Director of the Computer Systems Laboratory at Stanford. From 1979 to 1991 he was on the faculty of the Computer Science Department at Princeton University, Princeton, New Jersey. His research interests include distributed computing systems, digital libraries and database systems. He received a BS in electrical engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. From Stanford University, Stanford, California, he received in 1975 a MS in electrical engineering and a PhD in computer science in 1979. He holds an honorary PhD from ETH Zurich (2007). Garcia-Molina is a Fellow of the Association for Computing Machinery and of the American Academy of Arts and Sciences; is a member of the National Academy of Engineering; received the 1999 ACM SIGMOD Innovations Award; is a Venture Advisor for Onset Ventures, and is a member of the Board of Directors of Oracle.