Managing Information Leakage

Steven Euijong Whang and Hector Garcia-Molina
Computer Science Department
Stanford University
353 Serra Mall, Stanford, CA 94305, USA
{swhang, hector}@cs.stanford.edu

ABSTRACT

We explore the problem of managing information leakage by connecting two hitherto disconnected topics: entity resolution (ER) and data privacy (DP). As more of our sensitive data gets exposed to a variety of merchants, health care providers, employers, social sites and so on, there is a higher chance that an adversary can "connect the dots" and piece together our information, leading to even more loss of privacy. For instance, suppose that Alice has a social networking profile with her name and photo and a web homepage containing her name and address. An adversary Eve may be able to link the profile and homepage to connect the photo and address of Alice and thus glean more personal information. The better Eve is at linking the information, the more vulnerable is Alice's privacy. Thus in order to gain DP, one must try to prevent important bits of information being resolved by ER. In this paper, we formalize information leakage and list several challenges both in ER and DP. We also propose using disinformation as a tool for containing information leakage.

1. INTRODUCTION

In this paper we explore the connections between two hitherto disconnected topics: entity resolution and data privacy. In entity resolution (ER), one tries to identify data records that refer to the same real world entity. Matching records are often merged into "composite" records that reflect the aggregate information known about the entity. The goal of data privacy (DP) is to prevent disclosure to third parties of sensitive user (entity) data. For instance, sensitive data can be encrypted to make it hard for a third party to obtain, or the sensitive data can be "modified" (e.g., changing an age 24 to a range "between 20 and 30").

We will argue that in a sense ER and DP are opposites: the better one is at ER, them more one learns about real world entities, including their sensitive information. And to achieve DP, one must try to prevent the bits of information that have been published about an entity from being glued together by ER.

To illustrate, we present a simple motivating example. Consider an entity (person) Alice with the following information: her name is Alice, her address is 123 Main, her phone number is 555, her credit card number is 999, her social security number is 000. We represent Alice's information as the record: { ⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 555⟩, ⟨C, 999⟩, ⟨S, 000⟩ }. Suppose now that Alice buys something on the Web and gives the vendor a subset of her information, say {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨C, 999⟩}. By doing so, Alice has already partially compromised her privacy. We can quantify this "information leakage" in various ways: for instance we can say that the vendor has 3 out of 5 of Alice's attributes, hence the recall is $\frac{3}{5}$. We view leakage as a continuum, not as all-or-nothing. Low leakage (recall in our example metric) is desirable, since the vendor (or third party) knows less about Alice, hence we try to *minimize* leakage. (Note we can actually weight attributes in our leakage computation by their sensitivity.)

Next, say Alice gets a job, so she must give her employer the following data: {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 555⟩, ⟨S, 000⟩}. In this case the leakage is $\frac{4}{5}$. This is where ER comes into play: If the employer and vendor somehow pool their data, they may be able to figure out that both records refer to the same entity. In general, in ER there are no unique identifiers: one must analyze the data and see if there is enough evidence. In our example, say the common name and address (and the lack of conflicting information) imply that the records match and are combined (and say the attributes are unioned). Then the third party has increased leakage (recall) to 1.

If Alice wants to prevent this increase in leakage, she may release *disinformation*, e.g., a new record that prevents the resolution that increased leakage. For example, say that Alice somehow gives the vendor the following additional record: {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 666⟩, ⟨C, 999⟩}. (Note the incorrect phone number.) Now the vendor resolves this third record with its first record, reaching the conclusion that Alice's phone number is 666. Now, when the vendor and employer pool their information, the different phone numbers lead them to believe that records correspond to different entities, so they are not merged and leakage does not increase. The incorrect phone number also decreases another metric we will consider, precision, since now not all of the third party's data is correct. Thus, leakage can decrease, not just by knowing less about Alice, but by mixing the correct data about Alice with incorrect data.

Before proceeding with the main body of our paper, we

5^{th} *Biennial Conference on Innovative Data Systems Research (CIDR '11)* January 9-12, 2011, Asilomar, California, USA.

highlight the key features of our approach, which we believe make it well suited for studying ER and DP:

- Although not illustrated in our example, our model captures data confidences and multiple attribute values, both which arise naturally in ER. In particular, we believe less information has leaked if a third party is uncertain about Alice's attributes, as opposed to the case where the third party is certain.
- In most DP work, privacy is all-or-nothing, while as mentioned above, our leakage ranges between 0 (no information known by third party) to 1 (all information, and only correct information, is known). We believe that our continuous leakage model is more appropriate in our case: Alice *must* give out some sensitive data in order to buy products, get jobs, and so on. We cannot guarantee full privacy in this context; we can only quantify (and hopefully minimize) leakage. Furthermore, we are able to capture the notion that more leaked attributes is worse than fewer. For example, if a third party only knows our credit card number, that by itself is not a great loss. If the third party also learns our card expiration date, that is a more serious breach. If in addition they know our name and address, the information leakage is more serious.
- So far we have phrased leakage as a bad thing, something to be minimized. However, our model can also be used to study the mirror problem, where a good analyst is using ER to discover information about "bad guys". Here the goal is to maximize leakage, i.e., to discover how to perform ER so we can learn the most correct information about adversaries.

As we will show, our framework can help us answer fundamental questions on information leakage, for instance:

- Alice needs to give certain information to a store. She may want to know the impact of this release: Will the new information allow the store to "connect the dots" and piece together many previously released records? Or will the leakage increase be minimal?
- An analyst may want to understand what ER algorithms are best to increase information leakage.
- Alice may want to use disinformation to reduce the impact of previously leaked information. By adding some bogus information about herself, it becomes more costly for Eve to resolve Alice's correct information.

In this short paper we present a relatively brief summary of our work on the convergence of ER and DP. Our technical report [5] contains full details. In a nutshell, our contributions are two-fold:

(1) We propose a framework for measuring information leakage (summarized in Section 2 of this paper), and

(2) We study how the framework can be used to answer a variety of questions related to leakage and entity resolution. Section 3 of this paper briefly describes two of the questions we have studied (exemplified by the first and third bullets immediately above).

2. MODELS AND ALGORITHMS

We assume a database of records $R = \{r_1, r_2, \ldots, r_n\}$. The database could be a collection of social networking profiles, homepages, or even tweets. Or we can also think of R as a list of customer records of a company. Each record r is a set of attributes, and each attribute consists of a label and value. (In Section 2.4 we extend the model to values with confidences.) We do not assume a fixed schema because records can be from various data sources that use different attributes. As an example, the following record may represent Alice:

$$r = \{\langle \text{N, Alice}\rangle, \langle \text{A}, 20\rangle, \langle \text{A}, 30\rangle, \langle \text{Z}, 94305\rangle\}$$

Each attribute $a \in r$ is surrounded by angle brackets and consists of one label $a.lab$ and one value $a.val$. Notice that there are two ages for Alice. We consider ⟨A, 20⟩ and ⟨A, 30⟩ to be two separate pieces of information, even if they have the same label. Multiple label-value pairs with identical labels can occur when two records combine and the label-value pairs are simply collected. In our example, Alice may have reported her age to be 20 in some case, but 30 in others. (Equivalently, year of birth can be used instead of age.) Although we cannot express the fact that Alice has only one age (either 20 or 30), the confidences we introduce in Section 2.4 can be used to indicate the likelihood of each value.

2.1 Record Leakage

We consider the scenario where Eve has one record r of Alice in her database R. (We consider the case where R contains multiple records in Section 2.2.) In this scenario, we only need to measure the information leaked by r in comparison to the "reference" record p that contains the complete information of Alice. We define $L_r(r, p)$ as the *record leakage* of r against p.

While leakage can be measured in a variety of ways, we believe that the well known concepts of precision and recall (and the corresponding F_1 metric) are very natural for this task. We first define the precision Pr of the record r against the reference p as $\frac{|r \cap p|}{|r|}$. Intuitively, Pr is the fraction of attributes in r that are also correct according to p. Suppose that $p = \{\langle \text{N, Alice}\rangle, \langle \text{A, 20}\rangle, \langle \text{P, 123}\rangle, \langle \text{Z, 94305}\rangle\}$ and $r = \{\langle \text{N, Alice}\rangle, \langle \text{A, 20}\rangle, \langle \text{P, 111}\rangle\}$. Then the precision of r against p is $\frac{2}{3} \approx 0.67$. We next define the recall Re of r against p as $\frac{|r \cap p|}{|p|}$. The recall reflects the fraction of attributes in p that are also found in r. In our example, the recall of r against p is $\frac{2}{4} = 0.5$. We can combine the precision and recall to produce a single metric called $F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$. In our example, the F_1 value is $\frac{2 \times 0.67 \times 0.5}{0.67 + 0.5} \approx 0.57$. It is straightforward to extend our definitions of precision and recall to weighted attributes, where the weight of an attribute reflects its sensitivity.

2.2 Query Leakage

We now consider the case where Eve has a database R containing multiple records. These records can represent information on different entities, and can be obtained directly from the entities, from public sources, or from other organizations Eve pools information with.

When Eve had a single record (previous section), we implicitly assumed that the one record was about Alice and we computed the resulting leakage based on what Eve knew about Alice. Now with multiple records, how does Eve know which records are "about Alice" and leak Alice's information? And what happens if multiple database records are about Alice?

To address these questions, we now define leakage, not as an absolute, but relative to a "query". For instance, Eve

Rec.	Type	Attributes
r_1	Social	⟨N, Alice⟩ , ⟨P, 123⟩ , ⟨B, Jan. 10⟩
r_2	Homepage 1	⟨N, Alice⟩ , ⟨C, Google⟩, ⟨A, 30⟩
r_3	Homepage 2	⟨N, Alice⟩, ⟨E, Stanford⟩ , ⟨A, 20⟩
r_4	Homepage 3	⟨N, Alice⟩, ⟨C, Boggle⟩, ⟨A, 50⟩

Table 1: Records of Alice on the Web

may pose the query "What do I know about {⟨N, Alice⟩, ⟨A, 123Main⟩}". In this case, Eve is saying that the attributes "name: Alice" and "address: 123Main" identify an entity of interest to her, and would like to know what else is known about this entity. Note that this pair of attributes is not necessarily a unique key that identifies entity Alice; the two attributes are simply how Eve thinks of entity Alice. They may be insufficient to uniquely identify Alice, they may be more than is needed. Furthermore, there could be different attributes that also identify Alice.

Our next step is to compute leakage (relative to this query) by figuring out what Eve knows related to {⟨N, Alice⟩, ⟨A, 123Main⟩}. But which database records are related to this query? And how are all the related records combined into what Eve knows about Alice?

To answer these questions, we introduce what we call the *match* and the *merge* functions in ER. A match function M compares two records r and s and returns true if r and s refer to the same real-world entity and false otherwise. A merge function μ takes two matching records r and s and combines them into a single record $\mu(r, s)$.

To illustrate how we use these functions to evaluate leakage, consider the database of Table 1, owned by Eve. Suppose that Eve identifies Alice by the query q={⟨N, Alice⟩, ⟨C, Google⟩} (i.e., the Alice that works at Google). What else does Eve know about this Alice? We use a process called *dipping* to discover database records that match (defined by our function) the query record. That is, we first look for a database record that matches the query. When we find it, we merge the matching record with the query, using our merge function. In our example, say record r_2 matches q, so we obtain $r_q = \mu(q, r_2)$. Then we look for any database record that matches r_q, the expanded query, and merge it to our expanded record. For instance, say $M(r_q, r_1)$ evaluates to true, so we replace r_q by $\mu(r_q, r_1)$. We continue until no other database record matches. This process is called dipping because it is analogous to dipping say a pretzel (the query) into a vat of melted chocolate (the database). Each time we dip the pretzel, more and more chocolate may adhere to the pretzel, resulting in a delicious chocolate-covered pretzel (or an expanded query with all information related to Alice). Note that dipping is a type of entity resolution, where records in the database match against one record (the pretzel), as opposed to any record in the database.

At the end of the dipping process, r_q represents what Eve knows about Alice (q), so we evaluate the leakage by comparing r_q to Alice's private information p, as before. Note that we will get different leakage for different queries. For instance, if Eve thinks of Alice as q={⟨N, Alice⟩}, more records will conglomerate in our example, which may lead to lower leakage (if r_3 and r_4 actually refer to different Alices) or higher leakage (if r_3 and r_4 are the same Alice as r_1 and r_2).

In the remainder of this section we define the dipping process more formally. We start by defining two properties that match and merge functions generally have (and that we assume for our work).

We assume two basic properties for M and μ – commutativity and associativity. Commutativity says that, if r matches s, then s matches r as well. In addition, the merged result of r and s should be identical regardless of the merge order. Associativity says that the merge order is irrelevant.

- Commutativity: $\forall r, s, M(r, s) =$ true if and only if $M(s, r)$ = true, and if $M(r, s) =$ true, $\mu(r, s) = \mu(s, r)$
- Associativity: $\forall r, s, t$, $\mu(r, \mu(s, t)) = \mu(\mu(r, s), t)$

We believe that most match and merge functions will naturally satisfy these properties. Even if they do not, they can easily be modified to satisfy the properties. To illustrate the second point, suppose that commutativity does not hold because $M(r, s)$ only compares r and s if r has an age smaller or equal to s and returns false otherwise. In that case, we can define the new match function $M'(r, s)$ to invoke $M(r, s)$ if r's age is smaller or equal to s's age and invoke $M(s, r)$ if s's age is smaller than r. In the case where the two properties are not satisfied, we only need to add a few more lines in our dipping algorithms for correctness.

Before defining the dipping result of a set of records, we define the "answer sets" for Alice. Throughout the paper, we use the short-hand notation $\mu(S)$ for any associative merge function μ as the merged result of all records in the set of records S (if $S = \{r\}$, $\mu(S) = r$).

DEFINITION 2.1. *Given a query q, a match function M, a merge function μ, and a set of record R, the collection of* answer sets *is $A = \{S_1, \ldots, S_m\}$ where each $S_i \in A$ is a set of records that satisfies the following conditions.*

- $q \in S_i$
- $S_i \subseteq R \cup \{q\}$
- *The records in $S_i - \{q\}$ can be reordered into a sequence $[r_1, \ldots, r_m]$ such that $M(q, r_1) =$ true, $M(\mu(q, r_1), r_2) =$ true, $\ldots$, $M(\mu(\{q, r_1, \ldots, r_{m-1}\}), r_m) =$ true*

For example, suppose we have a database $R = \{r_1, r_2\}$ where r_1 and r_2 are clearly not the same person and have the names Alice and Alicia, respectively. However, say the query q matches either r_1 or r_2 because it contains both names Alice and Alicia and no other information. Then the answer set is $A = \{\{\}, \{q, r_1\}, \{q, r_2\}\}$. Notice that in this example the set $\{q, r_1, r_2\}$ is not in A because r_1, even after it merges with q, does not match r_2.

A dipping result of R is then the merged result of a "maximal" answer set from Definition 2.1 that has no other matching record in R.

DEFINITION 2.2. *Given the collection of answer sets A, a match function M, and a merge function μ, $r_q = \mu(S)$ is a* dipping result *if $S \in A$ and $\forall r \in R - S$, $M(r, \mu(S)) =$ false.*

Continuing our example from above, the dipping result r_q can be either $\mu(r_1, q)$ or $\mu(r_2, q)$ because once q merges with r_1 (r_2), it cannot merge with r_2 (r_1). Notice that we exclude the case of merging multiple records with r_q at a time. For example, even if r_q matches with the merged record $\mu(r, s)$, but not with r or s individually, then we still cannot merge r_q with r and s.

While Definition 2.2 assumes that one record is added to q at a time, it can easily be extended to capture more

sophisticated dipping such as adding multiple records to q at a time.

We define the *query leakage* $L_q(p, q, M, \mu, R)$ of Alice as the maximum value of $L_r(p, r_q)$ for all possible dipping results r_q that can be produced using the match and merge functions M and μ on the database R. In general, deriving the query leakage is an NP-hard problem (see our technical report [5] for a proof).

Properties. We identify two desirable properties for M and μ: representativity and negative representativity. Representativity says that a merged record $\mu(r, s)$ "represents" r and s and matches with all records that match with either r or s. Intuitively, there is no "negative evidence" so merging r and s cannot create evidence that would prevent $\mu(r, s)$ from matching with any record that matches with r or s. It can be shown that representativity guarantees the uniqueness of a dipping result and is needed for efficient dipping. Negative representativity says that two records r and s that do not match will never match even if r or s merges with other records. That is, there is no "positive evidence" where r and s will turn out to be the same entity later on. The negative representativity property also enables efficient dipping. Note that the two properties above do not assume that r matches with s. We can show that none of the properties imply each other.

- Representativity: If $t = \mu(r, s)$, then for any u where $M(r, u) =$ true, we also have $M(t, u) =$ true
- Negative Representativity: If $t = \mu(r, s)$, then for any u where $M(r, u) =$ false, we also have $M(t, u) =$ false

We illustrate a match function called M_c and merge function called μ_u that satisfy both representativity and negative representativity (the proof that M_c and μ_u satisfy the properties and more examples of match and merge functions can be found in our technical report [5]). The M_c function uses a single "key set" for comparing two records. A key set k is a minimal set of attribute labels $\{l_1, \ldots, l_m\}$ that are sufficient to determine if two records are the same using equality checks. All records are assumed to have values for the key-set attributes. The M_c function then matches r and s only if they have the exact same key-set attributes. For example, given the key set $k = \{A, B\}$, the record $r = \{\langle A, a\rangle, \langle B, b\rangle\}$ matches with $s = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c\rangle\}$, but not with $t = \{\langle A, a\rangle, \langle A, a'\rangle, \langle B, b\rangle\}$. As another example, the record $r = \{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$ matches with $s = \{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$, but not with $t = \{\langle A, a_1\rangle, \langle B, b\rangle\}$. The merge function μ_u unions the attributes of r and s (i.e., $\mu(r, s) = r \cup s$). For instance, if $r = \{\langle A, a\rangle, \langle B, b\rangle\}$ and $s = \{\langle A, a\rangle, \langle C, c\rangle\}$, then $\mu(r, s) = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c\rangle\}$.

Dipping Algorithms. In our technical report [5], we explore various dipping algorithms that exploit properties. (Table 2 in Section 2.5 summarizes their complexities.)

2.3 Database Leakage

What happens if we do not know how Eve identifies Alice, i.e., if we do not have a specific query q? In such a case, we may assume that Eve can think of any one of the records in R as "Alice's record". Thus, for each R record we can compute a leakage number, and by taking the maximum value we can obtain a worst case leakage, representing what Eve can *potentially* know about Alice.

More formally, we define the database leakage $L_d(p, M, \mu, R)$ as $max_{q \in R}\ L_q(p, q, M, \mu, R - \{q\})$. That is, for each record $q \in R$, we compute the dipping of q on $R - \{q\}$ (i.e., the database without q) and choose the worst-case query leakage of Alice as the entire database leakage. In our technical report [5], we explore various algorithms that compute the database leakage of R (Table 2 in Section 2.5 summarizes their complexities) as well as techniques to further scale the algorithms.

2.4 Uncertain Data

As we argued in the introduction, data confidence plays an important role in leakage. For instance, Eve "knows more about Alice" if she is absolutely sure Alice is 50 years old (correct value), as opposed to thinking she might be 50 years old with say 30% confidence, or thinking Alice is either 30 or 50 years old. To capture this intuition, we extend our model to include uncertain data values. Note that there are many ways to model data uncertainty, and our goal here is not to use the most sophisticated model possible. Rather, our goal is to pick a simple uncertainty model that is sufficient for us to study the interaction between uncertainty and information leakage.

Thus, in our extended model, each record r in R consists of a set of attributes, and each attribute contains a label, a value, and a confidence (from 0 to 1) that captures the uncertainty of the attribute (from Eve's point of view). Any attribute that does not exist in r is assumed to have a confidence of 0. As an example, the following record may represent Alice:

$$r = \{\langle \text{N}, \text{Alice}, 1\rangle, \langle \text{A}, 20, 0.5\rangle, \langle \text{A}, 30, 0.4\rangle, \langle \text{Z}, 94305, 0.3\rangle\}$$

That is, Eve is certain about Alice's name and age, but is only 50% confident about Alice being 30 years old, 40% confident in Alice being 30 years old, and 30% confident about Alice's zip code 94305. For each attribute $a \in r$, we can access a's label $a.lab$, a single value $a.val$, and confidence $a.cnf$. In Table 1, the outdated record r_3 of Alice can be viewed to have a lower confidence than the up-to-date record r_2. We assume that attributes in the reference p always have a confidence of 1. We require that no two attributes in the same record can have the same label and value pair.

The confidences within the same record are independent of each other and reflect "alternate worlds" for Eve's belief of the correct Alice information. For example, if we have $r = \{\langle$name, Alice, 1$\rangle$, $\langle$age, 20, 0.5$\rangle$, $\langle$phone, 123, 0.5$\rangle\}$, then there are four possible alternate worlds for r with equal probability: $\{\langle$name, Alice$\rangle\}$, $\{\langle$name, Alice$\rangle$, $\langle$age, 20$\rangle\}$, $\{\langle$name, Alice$\rangle$, $\langle$phone, 123$\rangle\}$, and $\{\langle$name, Alice$\rangle$, $\langle$age, 20$\rangle$, $\langle$phone, 123$\rangle\}$.

We show one example on how our record leakage metrics in Section 2.1 can be extended to use confidences. We first define the notation $IN(r, s)$ for two records r and s as $\{a \in r \mid \exists a' \in s \text{ s.t. } a.lab = a'.lab \wedge a.val = a'.val\}$. That is, $IN(r, s)$ denotes the attributes of r whose label-value pairs exist in s. The precision Pr or a record r against the reference p is defined as $\frac{\Sigma_{t \in IN(r,p)} t.cnf}{\Sigma_{t \in r} t.cnf}$. Compared to the previous definition in Section 2.1, we now sum the confidences of the attributes in r whose label-value pairs are also in p and divide the result by the total confidence of r. The recall Re of r against p is defined as $\frac{\Sigma_{t \in (r \cap p)} t.cnf}{\Sigma_{t \in p} t.cnf}$. This time, we divide the sum of confidences of the attributes in r whose

Confidence	Properties	Query	Database
No	(none)	NP-hard	NP-hard
No	Representativity	$O(N^2)$	$O(N^3)$
No	Neg. Representativity Representativity	$O(N)$	$O(N^2)$
Yes	(none)	NP-hard	NP-hard
Yes	Representativity Monotonicity Increasing	$O(N^2)$	$O(N^3)$
Yes	Neg. Representativity Representativity Monotonicity Increasing	$O(N)$	$O(N^2)$

Table 2: Summary of Leakage Measurements

label-value pairs are also in p by the total confidence of p. We define the record leakage L_r as the F_1 metric $\frac{2\times Pr\times Re}{Pr+Re}$.

In our technical report [5], we elaborate on how to extend our leakage algorithms to take into account confidences. In addition, we discuss two properties (called *monotonicity* and *increasing*) for the extended match and merge functions that can be exploited to compute leakage efficiently.

2.5 Summary of Contributions

As mentioned earlier, Table 2 summarizes the scenarios we have considered in our work. The first column shows whether or not the adversary (which we call Eve) uses confidences in the leakage model. The second column shows properties that are satisfied by the match and merge functions. For each scenario (row) we have developed an algorithm that computes either query or database leakage (given a reference record p for Alice, and a database R held by Eve), and columns 3 and 4 show the complexity of these algorithms. As one can see in the table, the more properties that hold, the more efficiently we can compute leakage.

The properties depend on the data semantics, and different applications (e.g., commercial products, publications, personal information) will have different properties. To show that the properties are achievable in practice, for each scenario in Table 2 we have developed simple match and merge functions that satisfy the corresponding properties. These functions, as well as the algorithms, are all detailed in our technical report [5].

3. USING OUR FRAMEWORK

Our framework can be used to answer a variety of questions, and here we illustrate two questions. As we use our framework, it is important to keep in mind "who knows what". In particular, if Alice is studying leakage of her information (as in the two examples we present here), she needs to make assumptions as to what her adversary Eve knows (database R) and how she operates (the match and merge functions, and dipping algorithm Eve uses). These types of assumptions are common in privacy work, where one must guess the sophistication and compute power of an adversary. On the other hand, if Eve is studying leakage she will not have Alice's reference information p. However, she may use a "training data set" for known individuals in order to tune her dipping algorithms, or say estimate how much she really knows about Alice.

3.1 Releasing Critical Information

Suppose that Alice wants to purchase a cellphone app from one of two stores S_1 and S_2, and is wondering which purchase will lead to a more significant loss of privacy. Both stores require Alice to submit her name, credit card number, and phone number for the app. However, due to Alice's previous purchases, each store has different information about Alice. In particular:

- Alice's reference information is $p = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{P}, p_1, 1\rangle, \langle \text{A}, a_1, 1\rangle\}$ where N stands for name, C for credit card number, P for phone, and A for address.
- Store S_1 has one previous record $R_1 = \{r = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{A}, a_1, 1\rangle\}\}$. That is, Alice bought an item using her credit card number and shipping address. (We omit the item information in any record for brevity.)
- Store S_2 has two previous records $R_2 = \{s = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{P}, p_1, 1\rangle\}, t = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{A}, a_1, 1\rangle\}\}$. Here, Alice has bought items using different credit cards. The item of s could be a ringtone that required a phone number for purchasing, but not a shipping address.
- Both S_1 and S_2 require the information $u = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{P}, p_1, 1\rangle\}$ for the cellphone app purchase. Since Alice is purchasing an app, again no shipping address is required.

To compute leakages, say Alice is only concerned with the previously released information, so she assumes that the database at store S_1 only contains record r, while the database at store S_2 only contains s and t. (The stores are not colluding in this example.) Alice also assumes that two records match if their names and credit card numbers are the same or their names and phone numbers are the same, and that merging records simply performs a union of attributes.

Under these assumptions, before Alice's app purchase, the database leakage for both stores is $\frac{3}{4}$. For the first store, R_1 only contains one record r, so the database leakage is $L_r(p,r) = \frac{2\times 1\times 3/5}{1+3/5} = \frac{3}{4}$. For the second store, R_2 contains two records s and t, so we need to take the maximum of the query leakages of s and t. Since s and t do not match with each other (i.e., they do not have the same name and credit card or name and phone combination), the dipping result of s is s while the dipping result of t is t. Hence, the database leakage is $max\{L_r(p,s), L_r(p,t)\} = max\{\frac{3}{4}, \frac{3}{4}\} = \frac{3}{4}$.

If Alice buys her app from S_1, then its database will contain two records, r and u. In this case, the database leakage at S_1 is still $\frac{3}{4}$ because r and u do not match and thus have the same query leakage $\frac{3}{4}$ (the maximum query leakage is thus $\frac{3}{4}$). On the other hand, if Alice buys from S_2, the database at S_2 will contain s, t, and u. Since u matches with both s and t, the dipping result of u is $\mu(\{s,t,u\})$, which is identical to p. Hence, the database leakage becomes 1.

To compare Alice's two choices, it is useful to think of the *incremental leakage*, that is, the change in leakage due to the app purchase. In our example, the incremental leakage at S_1 is $\frac{3}{4} - \frac{3}{4} = 0$ while the incremental leakage at S_2 is $1 - \frac{3}{4} = \frac{1}{4}$. Thus, in this case Alice should buy her app from S_1 because it preserves more of her privacy.

3.2 Releasing Disinformation

Given previously released information R, a match function M, and a merge function μ, Alice may want to release

either a single record or multiple records that can decrease the query or database leakage. We call records that are used to decrease the database leakage *disinformation* [1] records. Of course, Alice can reduce the query or database leakage by releasing arbitrarily large disinformation. However, disinformation itself has a cost. For instance, adding a new social network profile would require the cost for registering information. As another example, longer records could require more cost and effort to construct. We use $C(r)$ to denote the entire cost of creating r.

We define the problem of minimizing the database leakage using one or more disinformation records. Given a set of disinformation records S and a maximum budget of C_{max}, the optimal disinformation problem can be stated as the minimization function presented below:

$$\begin{array}{ll} \text{minimize} & L_d(p, M, \mu, R \cup S) \\ \text{subject to} & \Sigma_{r \in S} C(r) \leq C_{max} \end{array}$$

The problem of minimizing the query leakage can also be stated by replacing L_d by L_q in the above formula. The set of records S that minimizes the database leakage within our cost budget C_{max} is called "optimal" disinformation.

A disinformation record r_d can reduce the database leakage in two ways. First, r_d can perform *self disinformation* by directly adding the irrelevant information it contains to the dipping result r_q that yields the maximum query leakage. For example, given the database $R = \{r, s, t\}$ and a reference p, suppose the database leakage is $L_r(p, \mu(r, s))$. Then r_q can be created to match with $\mu(r, s)$ and to contain bogus data not found in p, thus decreasing database leakage to $L_r(p, \mu(\{r, s, r_d\}))$. Second, r_d can perform *linkage disinformation* by linking irrelevant records in R to r_q. For example, say that t contains totally irrelevant information of the target p. If r_d can be made to match with both $\mu(r, s)$ and t, then the database leakage could decrease to $L_r(p, \mu(\{r, s, t, r_d\}))$ because of r_d. Of course, r_d can also use both self and linkage disinformation.

When creating a record, we use a user-defined function called $Create(S, L)$ that creates a new minimal record that has a size less or equal to L and is guaranteed to match all the records in the set S. If there is no record r such that $|r| \leq L$ and all records in S match with r, the $Create$ function returns the empty record $\{\}$. A reasonable assumption is that the size of the record produced by $Create$ is proportional to $|S|$ when $L > |S|$. We also assume a function called $Add(r)$ that appends a new attribute to r. The new attribute should be "incorrect but believable" (i.e., bogus) information. We assume that if two records r and s match, they will still match even if Add appends bogus attributes to either r or s. (Notice that this property is similar to the representativity property for match and merge functions.) The $Create$ function is assumed to have a time complexity of $O(|S|)$ while the Add function $O(|r|)$. In our technical report [5], we list more details to consider when adding bogus attributes to r_d using the function Add.

We propose disinformation algorithms in our technical report [5], both for the case we release a single disinformation record (S is size 1) and the case where we release multiple records. In addition, we consider scenarios where different properties hold. If both representativity and negative representativity hold, one can show that a new record can only use self-disinformation to lower the database leakage, which enables efficient algorithms that return the optimal disinformation. If the properties do not hold, a new record can also use linkage disinformation, and we might have to consider all possible combinations of irrelevant records in the worst case (which makes the disinformation problem NP-hard). Thus, we also propose a heuristic algorithm that searches a smaller space, where we either combine two irrelevant records and use self disinformation or use self disinformation only. As more properties are satisfied by the match and merge functions, the more efficient the disinformation algorithms become. Similar results can be obtained when using confidences and the monotonicity and increasing properties for the extended match and merge functions.

[1] A classic example of disinformation occurred before the Normandy landings during World War II where British intelligence convinced the German Armed Forces that a much larger invasion was about to cross the English Channel from Kent, England.

4. RELATED WORK

Many works have proposed privacy schemes for data publishing in the context of linkage attacks. Various models including k-anonymity [3] and l-diversity [1] guarantee that linkage attacks on certain attributes cannot succeed. In contrast, we assume that the data is already published and that we want to manage the leakage of sensitive information.

Several closely related products manage information leakage. A service called ReputationDefender [2] manages the reputation of individuals, e.g., making sure a person's correct information appears on top search results. TrackMeNot [4] is a browser extension that helps protect web searchers from surveillance and data-profiling by search engines using noise and obfuscation. In comparison, our work complements the above works by formalizing information leakage and proposing disinformation as a general tool for containing leakage.

5. CONCLUSION

We have proposed a framework for managing information leakage and studied how the framework can be used to answer a variety of questions related to ER and DP. The algorithms for computing leakage become more efficient as the match and merge functions satisfy more properties. We have studied the problems of measuring the incremental leakage of critical information and using disinformation as a tool for containing information leakage. We believe our techniques are preliminary steps to the final goal of truly managing public data, and that many interesting problems remain to be solved.

6. REFERENCES

[1] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

[2] ReputationDefender. http://www.reputationdefender.com.

[3] L. Sweeney. k-anonymity: A model for protecting privacy. *IJUFKS*, 10(5):557–570, 2002.

[4] TrackMeNot. http://cs.nyu.edu/trackmenot.

[5] S. E. Whang and H. Garcia-Molina. Managing information leakage. Technical report, Stanford University, available at http://ilpubs.stanford.edu:8090/983/.