Anonymizing Sequential Releases [*]

Ke Wang
School of Computing Science
Simon Fraser University
Canada V5A 1S6
wangk@cs.sfu.ca

Benjamin C. M. Fung
School of Computing Science
Simon Fraser University
Canada V5A 1S6
bfung@cs.sfu.ca

ABSTRACT

An organization makes a new release as new information become available, releases a tailored view for each data request, releases sensitive information and identifying information separately. The availability of related releases sharpens the identification of individuals by a global quasi-identifier consisting of attributes from related releases. Since it is not an option to anonymize previously released data, the current release must be anonymized to ensure that a global quasi-identifier is not effective for identification. In this paper, we study the *sequential anonymization* problem under this assumption. A key question is how to anonymize the current release so that it cannot be linked to previous releases yet remains useful for its own release purpose. We introduce the *lossy join*, a negative property in relational database design, as a way to hide the join relationship among releases, and propose a scalable and practical solution.

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: [Security, integrity, and protection]; H.2.8 [**Database Applications**]: [Data mining]

General Terms

Algorithms, Performance, Security

Keywords

k-anonymity, privacy, sequential release, classification, generalization

1. INTRODUCTION

The work on *k-anonymity* [16][17] addresses the problem of reducing the risk of identifying individuals in a person-specific table.

Typically, a set of identifying attributes in a table, called the *quasi-identifier* or QID, is generalized to a less precise representation so that each partition grouped by QID contains at least k records (i.e., persons). Hence, if some record is linked to an external source by a QID value, so are at least $k-1$ other records having the same QID value, making it difficult to distinguish a particular individual. In this notion, the QID is restricted to the current table, and *the database is made anonymous to itself*. In most scenarios, however, related data were released previously: an organization makes a new release as new information becomes available, releases a separate view for each data sharing purpose (such as classifying a different target variable [6][23][5]), or makes separate releases for personally-identifiable data (e.g., names) and sensitive data (e.g., DNA sequences) [11]. In such scenarios, the QID can be a combination of attributes from several releases, and *the database must be made anonymous to the combination of all releases thus far*. The example below illustrates this scenario.

1.1 Motivating Examples

Table 1: The join of T_1 and T_2

T_1				T_2		
Pid	Name	Job	Class	Pid	Job	Disease
1	Alice	Banker	c1	1	Banker	Cancer
2	Alice	Banker	c1	2	Banker	Cancer
3	Bob	Clerk	c2	3	Clerk	HIV
4	Bob	Driver	c3	4	Driver	Cancer
5	Cathy	Engineer	c4	5	Engineer	HIV

The join on $T_1.Job = T_2.Job$				
Pid	Name	Job	Disease	Class
1	Alice	Banker	Cancer	c1
2	Alice	Banker	Cancer	c1
3	Bob	Clerk	HIV	c2
4	Bob	Driver	Cancer	c3
5	Cathy	Engineer	HIV	c4
-	Alice	Banker	Cancer	c1
-	Alice	Banker	Cancer	c1

EXAMPLE 1. Consider the data in Table 1. Pid is the person identifier and is included only for discussion, not for release. Suppose the data holder has *previously* released T_2 and *now* wants to release T_1 for classification analysis of the $Class$ column. Essentially T_1 and T_2 are are two projection views of the patient records. The data holder does not want $Name$ to be linked to $Disease$ in the join of the two releases; in other words, the join should be k-anonymous on $\{Name, Disease\}$. Below are several observations that motivate our approach.

(1) ***Join sharpens identification***: after the join, the attacker can uniquely identify the individuals in the $\{Bob, HIV\}$ group through the combination $\{Name, Disease\}$ because this group has size 1. When T_1 and T_2 are examined separately, both Bob group and HIV group have size 2. (2) ***Join weakens identification***: after the join, the $\{Alice, Cancer\}$ group has size 4 because the records for different persons are matched (i.e., the last two records in the join table). When T_1 and T_2 are examined separately, both $Alice$ group and $Cancer$ group have smaller size. In the database terminology, the join is *lossy*. Since the join attack depends on matching the records for the *same* person, a lossy join can be used to combat the join attack. (3) ***Join enables inferences across tables***: the join reveals the inference $Alice \rightarrow Cancer$ with 100% confidence for the individuals in the $Alice$ group. ∎

This example illustrates a scenario of *sequential release*: T_1 was unknown when T_2 was released, and T_2, once released, cannot be modified when T_1 is considered for release. This scenario is different from the *view release* in the literature [13][26][7] where both T_2 and T_1 are a part of a view and can be modified before the release, which means more "rooms" to satisfy a privacy and information requirement. In the sequential release, each release has its own information need and the join that enables a global identifier should be prevented. In the view release, however, all tables in the view serve the information need collectively, possibly through the join of all tables.

One solution, suggested in [17], is to k-anonymize the current release T_1 on QID that is the set of all join attributes. Since a future release may contain any attribute in T_1, QID essentially needs to contain *all* attributes in T_1. Another solution, suggested in [19], is generalizing T_1 *based on* the previous T_2 to ensure that no value more specific than it appears in T_2 would be released in T_1. Both solutions suffer from monotonically distorting the data in a later release. The third solution is releasing a "complete" cohort where all potential releases are anonymized at one time, after which no additional mechanism is required. This solution requires predicting future releases. The "under-prediction" means no room for additional releases and the "over-prediction" means unnecessary data distortion. Also, this solution does not accommodate the new data added at a later time.

1.2 Contributions

We consider the *sequential anonymization* of the current release T_1 in the presence of a previous release T_2, assuming that T_1 and T_2 are projections of the same underlying table. This assumption holds in all the scenarios that motivate this work: release new attributes, release a separate set of columns for each data request, or make separate releases for personally-identifiable columns and sensitive columns. The release of T_1 must satisfy a given information requirement and privacy requirement. The information requirement could include such criteria as minimum classification error [2][5][6][23] and minimum data distortion [16][17]. The privacy requirement states that, even if the attacker joins T_1 with T_2, he/she will not succeed in linking individuals to sensitive properties. We formalize this requirement into limiting the linking between two attribute sets X and Y over the join of T_1 and T_2. This privacy notion, called (X, Y)-*privacy*, generalizes k-anonymity [16][17] and sensitive inferences [3][21][22]. A formal definition will be given in Section 3.

Our basic idea is generalizing the current release T_1 so that the join with the previous release T_2 becomes lossy enough to disorient the attacker. Essentially, a lossy join hides the true join relationship to cripple a global quasi-identifier. We first show that the sequential anonymization subsumes the k-anonymization, thus the optimal so-

lution is NP-hard. We present a greedy method for finding a minimally generalized T_1. To ensure the minimal generalization, the lossy join responds dynamically to each generalization step. Therefore, one challenge is checking the privacy violation over such dynamic join because a lossy join can be extremely large. Another challenge is pruning, as early as possible, unpromising generalization steps that lead to privacy violation. To address these challenges, we present a top-down approach to progressively specialize T_1 starting from the most generalized state. It checks the privacy violation without executing the join and prunes unpromising specialization based on a proven monotonicity of (X, Y)-privacy. We demonstrate the usefulness of this approach on real life data sets. Finally, we discuss the extension to more than one previous release.

2. RELATED WORK

Our major difference from previous works is that we consider sequential releases and a global quasi-identifier formed by attributes from several releases. Previous works primarily considered a single release. [1] [12] showed that the optimal k-anonymization is NP-hard. Algorithms for k-anonymization include [8][16][17] for minimum distortion, and [2][5][6][23] for classification. Variations and alternatives of k-anonymity were also studied. [9] proposed the notion of multidimensional k-anonymity where generalization is over multi-dimension-at-a-time. [10] proposed the l-diversity to address the attacks based on the lack of diversity of sensitive properties. [21][22] proposed to limit the confidence of inferring a sensitive property for a group of individuals. [24] proposed some generalization methods to simultaneously achieve k-anonymity and limit the confidence. [25] proposed the notion of personalized anonymity. All the above works considered a single release.

Several recent works measured information disclosure arising from linking two or more tables. [13] suggested a measure on information disclosure by a set of views with respect to a secret view. [4] studied whether a new view disclosed more information than the existing views with respect to a secret view. Both works employed a probability model to measure information disclosure, which is different from the k-anonymity model. [7][26] presented a method of detecting privacy violation by a view set over a base table. Since both works only detect, but do not remove, a violation, whether the tables are released sequentially or not is not an issue. [20] considered k-anonymization of the data owned by multiple parties under the assumption that a record is identified by a common key shared by all parties. In the sequential release scenario, this common key assumption does not hold and the join attributes can be generalized as part of the global quasi-identifier.

3. PROBLEM STATEMENTS

For a table T, $\Pi(T)$ and $\sigma(T)$ denote the projection and selection over T, $att(T)$ denotes the set of attributes in T, and $|T|$ denotes the number of distinct records in T.

3.1 Privacy

We assume that X and Y are disjoint sets of attributes that describe individuals and sensitive properties in any order. An example is $X = \{Name, Job\}$ and $Y = \{Disease\}$. There are two ways to limit the linking between X and Y.

DEFINITION 3.1 ((X, Y)-ANONYMITY). Let x be a value on X. The *anonymity* of x wrt Y, denoted $a_Y(x)$, is the number of distinct values on Y that co-occur with x, i.e., $|\Pi_Y \sigma_x(T)|$. If Y is a key in T, $a_Y(x)$, also written as $a(x)$, is equal to the number of records containing x. Let $A_Y(X) = min\{a_Y(x) \mid x \in X\}$.

We say that T *satisfies* the (X, Y)-anonymity for some specified integer k if $A_Y(X) \geq k$. ∎

In words, (X, Y)-anonymity states that each value on X is linked to at least k distinct values on Y. The existing k-anonymity is the special case where X serves QID and Y is a key in T. The next example shows the usefulness of (X, Y)-anonymity where Y is not a key in T and k-anonymity fails to provide the required anonymity.

EXAMPLE 2. Consider the table

$$Inpatient(Pid, Job, Zip, PoB, Test).$$

A record in the table represents that a patient identified by Pid has Job, Zip, PoB (place of birth), and $Test$. In general, a patient can have several tests, thus several records. Since $QID = \{Job, Zip, PoB\}$ is not a key in the table, the k-anonymity on QID fails to ensure that each value on QID is linked to at least k (*distinct*) patients. For example, if each patient has at least 3 tests, it is possible that the k records matching a value on QID may involve no more than $k/3$ patients. With (X, Y)-anonymity, we can specify the anonymity wrt *patients* by letting $X = \{Job, Zip, PoB\}$ and $Y = Pid$, that is, each X group must be linked to at least k distinct values on Pid. If $X = \{Job, Zip, PoB\}$ and $Y = Test$, each X group is required to be linked to at least k distinct tests. ∎

Being linked to k persons or tests does not imply that the probability of being linked to any of them is $1/k$ if some person or test occurs more frequently than others. Thus a large k does not necessarily limit the linking probability. The (X, Y)-linkability below addresses this issue.

DEFINITION 3.2 ((X, Y)-LINKABILITY). Let x be a value on X and y be a value on Y. The *linkability* of x to y, denoted $l_y(x)$, is the percentage of the records that contain both x and y among those that contain x, i.e., $a(y, x)/a(x)$. Let $L_y(X) = max\{l_y(x) \mid x \in X\}$ and $L_Y(X) = max\{L_y(X) \mid y \in Y\}$. We say that T *satisfies* the (X, Y)-linkability for some specified real $0 < k \leq 1$ if $L_Y(X) \leq k$. ∎

In words, (X, Y)-linkability limits the confidence of inferring a value on Y from a value on X. With X and Y describing individuals and sensitive properties, any such inference with a high confidence is a privacy breach. Often, not all but some values y on Y are sensitive, in which case Y can be replaced with a subset of y_i values on Y, written $Y = \{y_1, \cdots, y_p\}$, and a different threshold k can be specified for each y_i. More generally, we can allow multiple Y_i, each representing a subset of values on a different set of attributes, with Y being the union of all Y_i. For example, $Y_1 = \{HIV\}$ on $Test$ and $Y_2 = \{Banker\}$ on Job. Such a "value-level" specification provides a great flexibility essential for minimizing the data distortion.

EXAMPLE 3. Suppose that (j, z, p) on $X = \{Job, Zip, PoB\}$ occurs with the HIV test in 9 records and occurs with the $Diabetes$ test in 1 record. The confidence of $(j, z, p) \rightarrow HIV$ is 90%. With $Y = Test$, the (X, Y)-linkability states that no test can be inferred from a value on X with a confidence higher than a given threshold. ∎

When no distinction is necessary, we use the term "(X, Y)-privacy" to refer to either (X, Y)-anonymity or (X, Y)-linkability. The following corollary can be easily verified.

COROLLARY 3.1. Assume that $X \subseteq X'$ and $Y' \subseteq Y$. For the same threshold k, if (X', Y')-privacy is satisfied, (X, Y)-privacy is satisfied. ∎

3.2 Generalization/Specialization

One way to look at a (X, Y)-privacy is that Y serves the "reference point" with respect to which the privacy is measured. For example, with $Y = Test$ each test in Y serves a reference point, and $A_Y(X)$ measures the minimum number of tests associated with X, and $L_Y(X)$ measures the maximum confidence of inferring a test from X. To satisfy a (X, Y)-privacy, our approach is generalizing X while fixing the reference point Y. We assume that, for each categorical attribute in X, there is a pre-determined taxonomy tree of values where leaf nodes represent domain values and a parent node is a generalization of child nodes. The root is the most generalized value of the attribute, denoted ANY. Each *generalization* replaces all child values with the parent value. We consider only the generalization that forms a "cut" in a taxonomy tree, where a cut contains exactly one value on every root-to-leaf path. The values in a cut can be on different levels of the taxonomy tree. Such generalization is more general than the full-domain generalization [8][16][17] where all generalized values must be on the same level of the taxonomy tree.

A generalized table can be obtained by a sequence of specializations starting from the *most generalized table*. Each *specialization* is denoted by $v \rightarrow \{v_1, \cdots, v_c\}$, where v is the parent value and $v_1, \cdots, v_c$ are the child values of v. It replaces the value v in every record containing v with the child value v_i that is consistent with the original domain value in the record. A specialization for a continuous attribute has the form $v \rightarrow \{v_1, v_2\}$, where v_1 and v_2 are two sub-intervals of the larger interval v. Instead of being predetermined, the splitting point of the two sub-intervals is chosen on-the-fly to maximize information utility. More details on information utility will be discussed in Section 5.2.

3.3 Sequential Releases

Consider a previously released table T_2 and the current table T_1, where T_2 and T_1 are projections of the same underlying table and contain some common attributes. T_2 may have been generalized. We want to generalize T_1 to satisfy a given (X, Y)-privacy. To preserve information, T_1's generalization is not necessarily based on T_2, that is, T_1 may contain values more specific than in T_2. Given T_1 and T_2, the attacker may apply prior knowledge to match the records in T_1 and T_2. Entity matching has been studied in database, data mining, AI and Web communities for information integration, natural language processing and Semantic Web. We cannot consider a priori every possible way of matching. Our work primarily considers the matching based on the following prior knowledge available to both the data holder and the attacker: the schema information of T_1 and T_2, the taxonomies for categorical attributes, and the following inclusion-exclusion principle for matching the records. Assume that $t_1 \in T_1$ and $t_2 \in T_2$.

- *Consistency Predicate*: for every common categorial attribute A, $t_1.A$ matches $t_2.A$ if they are on the same generalization path in the taxonomy tree for A. Intuitively, this says that $t_1.A$ and $t_2.A$ can possibly be generalized from the same domain value. For example, $Male$ matches $Single\_Male$. This predicate is implicit in the taxonomies for categorical attributes.

- *Inconsistency Predicate*: for two distinct categorical attributes $T_1.A$ and $T_2.B$, $t_1.A$ matches $t_2.B$ only if $t_1.A$ and $t_2.B$ are not semantically inconsistent according to the "common sense". This predicate excludes impossible matches. If not specified, "not semantically inconsistent" is assumed. If two values are semantically inconsistent, so are their specialized values. For example, $Male$ and $Pregnant$ are semantically inconsistent, so are $Married\_Male$ and $6\_Month\_Pregnant$.

We do not consider continuous attributes because their taxonomies may be generated differently for T_1 and T_2. Both the data holder and the attacker use these predicates to match records from T_1 and T_2. The data holder can "catch up with" the attacker by incorporating the attacker's knowledge into such "common sense". We assume that a *match function* tests whether (t_1, t_2) is a match. (t_1, t_2) is a *match* if both predicates hold. The *join* of T_1 and T_2 is a table on $att(T_1) \cup att(T_2)$ that contains all matches (t_1, t_2). The *join attributes* refer to all attributes that occur in either predicates. Note that every common attribute A has two columns $T_1.A$ and $T_2.A$ in the join. The following observation says that generalizing the join attributes produces more matches, thereby making the join more lossy. Our approach exploits this property to hide the original matches.

Observation 1. (*Join preserving*) If (t_1, t_2) is a match and if t_1' is a generalization of t_1, (t_1', t_2) is a match. (*Join relaxing*) If (t_1, t_2) is not a match and if t_1' is a generalization of t_1 on some join attribute A, (t_1', t_2) is a match if and only if $t_1'.A$ and $t_2.A$ are on the same generalization path and $t_1'.A$ is not semantically inconsistent with any value in t_2.

Consider a (X, Y)-privacy. We generalize T_1 on the attributes $X \cap att(T_1)$, called the *generalization attributes*. Corollary 3.1 implies that including more attributes in X makes the privacy requirement stronger. Observation 1 implies that including more join attributes in X (for generalization) makes the join more lossy. Therefore, from the privacy point of view it is a good practice to include all join attributes in X for generalization. Moreover, if X contains a common attribute A from T_1 and T_2, under our matching predicate, one of $T_1.A$ and $T_2.A$ could be more specific (so reveal more information) than the other. To ensure privacy, X should contain both $T_1.A$ and $T_2.A$ in the (X, Y)-privacy specification.

DEFINITION 3.3 (SEQUENTIAL ANONYMIZATION). The data holder has previously released a table T_2 and wants to release the next table T_1, where T_2 and T_1 are projections of the same underlying table and contain some common attributes. The data holder wants to ensure a (X, Y)-privacy on the join of T_1 and T_2. The *sequential anonymization* is to generalize T_1 on $X \cap att(T_1)$ so that the join of T_1 and T_2 satisfies the privacy requirement and T_1 remains as useful as possible. ∎

THEOREM 3.1. The sequential anonymization is at least as hard as the k-anonymization problem.

Proof: The k-anonymization of T_1 on QID is the special case of sequential anonymization with (X, Y)-anonymity, where X is QID and Y is a common key of T_1 and T_2 and the only join attribute. In this case, the join trivially appends the attributes of T_2 to T_1 according to the common key, after which the appended attributes are ignored. ∎

4. MONOTONICITY OF PRIVACY

To generalize T_1, we will specialize T_1 starting from the most generalized state. A main reason for this approach is the following *anti-monotonicity* of (X, Y)-privacy with respect to specialization: if (X, Y)-privacy is violated, it remains violated after a specialization. Therefore, we can stop further specialization whenever the (X, Y)-privacy is violated for the first time. This is a highly desirable property for pruning unpromising specialization. We first show this property for a single table.

THEOREM 4.1. On a single table, the (X, Y)-privacy is anti-monotone wrt specialization on X.

Proof: For (X, Y)-anonymity, it suffices to observe that a specialization on X always reduces the set of records that contain a X value, therefore, reduces the set of Y values that co-occur with a X value. For (X, Y)-linkability, suppose that a specialization $v \to \{v_1, \cdots, v_c\}$ transforms a value x on X to the specialized values $x_1, \cdots, x_c$ on X. Following an idea in [21], if $l_y(x_i) < l_y(x)$ for some x_i, there must exist some x_j such that $l_y(x_j) > l_y(x)$ (otherwise, $l_y(x) < l_y(x)$). Hence, the specialization does not reduce $L_Y(X)$. ∎

On the join of T_1 and T_2, in general, (X, Y)-anonymity is not anti-monotone wrt a specialization on $X \cap att(T_1)$. To see this, let $T_1(C, D) = \{c_1d_3, c_2d\}$ and $T_2(D, Y) = \{d_3y_3, d_3y_2, d_1y_1\}$, where c_i, d_i, y_i are domain values and d is a generalized value of d_1 and d_2. The join based on D contains 3 matches (c_1d_3, d_3y_2), (c_1d_3, d_3y_3), (c_2d, d_1y_1), and $A_Y(X) = A_Y(c_2dd_1) = 1$, where $X = \{C, T_1.D, T_2.D\}$. After specializing the record c_2d in T_1 into c_2d_2, the join contains only two matches (c_1d_3, d_3y_2) and (c_1d_3, d_3y_3), and $A_Y(X) = a_Y(c_1d_3d_3) = 2$. Thus, $A_Y(X)$ increases after the specialization.

The above situation arises because the specialized record c_2d_2 matches no record in T_2 or becomes dangling. However, this situation does not arise for the T_1 and T_2 encountered in our sequential anonymization. We say that two tables are *population-related* if every record in each table has at least one matching record in the other table. Essentially, this property says that T_1 and T_2 are about the same "population" and there is no dangling record. Clearly, if T_1 and T_2 are projections of the same underlying table, as assumed in our problem setting, T_1 and T_2 are population-related. Observation 1 implies that generalizing T_1 preserves the population-relatedness.

Observation 2. If T_1 and T_2 are population-related, so are they after generalizing T_1.

LEMMA 4.1. If T_1 and T_2 are population-related, $A_Y(X)$ does not increase after a specialization of T_1 on $X \cap att(T_1)$.

Proof: As in the first part of Theorem 4.1, a specialization always reduces the set of Y values that co-occur with X values. From Observation 2, X values are specialized but not dropped in the specialized join. Therefore, the minimization for $A_Y(X)$ is over a set of values in which each value is only reduced, but not dropped. ∎

Now, we consider (X, Y)-linkability on the join of T_1 and T_2. It is not immediately clear how a specialization on $X \cap att(T_1)$ will affect $L_Y(X)$ because the specialization will reduce the matches, therefore, both $a(y, x)$ and $a(x)$ in $l_y(x) = a(y, x)/a(x)$. The next lemma shows that $L_Y(X)$ does not decrease after a specialization on $X \cap att(T_1)$.

LEMMA 4.2. If Y contains attributes from T_1 or T_2, but not from both, $L_Y(X)$ does not decrease after a specialization of T_1 on the attributes $X \cap att(T_1)$.

Proof: Theorem 4.1 has covered the specialization on a non-join attribute. So we assume that the specialization is on a join attribute in $X_1 = X \cap att(T_1)$, in particular, it specializes a value x_1 on X_1 into $x_{11}, \cdots, x_{1c}$. Let R_i be the set of T_1 records containing x_{1i} after the specialization, $1 \leq i \leq c$. We consider only non-empty R_i's. From Observation 2, some records in T_2 will match the records in R_i. Let x_{2i} be a value on $X_2 = X \cap att(T_2)$ in these matching records and let S_i be the set of records in T_2 containing x_{2i}. Note that $|R_i| \neq 0$ and $|S_i| \neq 0$. Let $R = R_1 \cup \cdots \cup R_c$. $|R| = \sum_j |R_j|$. Without loss of generality, assume that $l_y(x_{11}x_{21}) \geq l_y(x_{1i}x_{2i})$, where $1 \leq i \leq c$ and y is a Y value. We claim that $l_y(x_1x_{2i}) \leq l_y(x_{11}x_{21})$, which implies that the specialization does not decrease $L_y(X)$, therefore, $L_Y(X)$. The intuition is that of Theorem 4.1 and the insight that the join preserves

the relative frequency of y in all matching records. Let us consider two cases, depending on whether y is in T_1 or T_2.

Case 1: y is in T_1. Let σ_i be the percentage of the records containing y in R_i. Since all records in R_i match all records in S_i,

$$l_y(x_{1i}x_{2i}) = \frac{|R_i|\sigma_i|S_i|}{|R_i||S_i|} = \sigma_i.$$

From $l_y(x_{11}x_{21}) \geq l_y(x_{1i}x_{2i})$, we have $\sigma_1 \geq \sigma_i$, $1 < i \leq c$. From the join preserving property in Observation 1, all records in R match all records in S_i. So we have

$$l_y(x_1x_{2i}) = \frac{(\sum_j |R_j|\sigma_j)|S_i|}{|R||S_i|} = \frac{\sum_j |R_j|\sigma_j}{|R|} \leq \frac{\sigma_1 \sum_j |R_j|}{|R|}$$
$$= \sigma_1 = l_y(x_{11}x_{21}).$$

Case 2: y is in T_2. Let σ_i be the percentage of records containing y in S_i. Exactly as in Case 1, we can show $l_y(x_{1i}x_{2i}) = \sigma_i$ and $\sigma_1 \geq \sigma_i$, where $1 < i \leq c$, all records in R match all records in S_i. Now,

$$l_y(x_1x_{2i}) = \frac{|R||S_i|\sigma_i}{|R||S_i|} = \sigma_i \leq \sigma_1 = l_y(x_{11}x_{21}). \blacksquare$$

COROLLARY 4.1. *The (X, Y)-anonymity on the join of T_1 and T_2 is anti-monotone wrt a specialization of T_1 on $X \cap att(T_1)$. Assume that Y contains attributes from either T_1 or T_2, but not both. The (X, Y)-linkability on the join of T_1 and T_2 is anti-monotone wrt a specialization of T_1 on $X \cap att(T_1)$.* $\blacksquare$

COROLLARY 4.2. *Let T_1, T_2 and (X, Y)-privacy be as in Corollary 4.1. There exists a generalized T_1 that satisfies the (X, Y)-privacy if and only if the most generalized T_1 does.* $\blacksquare$

Remarks. Lemma 4.1 and Lemma 4.2 can be extended to several previous releases $T_2, \cdots, T_p$ after the join is so extended. Thus, the anti-monotonicity of (X, Y)-privacy holds for one or more previous releases. Our extension in Section 7 makes use of this observation.

5. ALGORITHMS

We present the algorithm for generalizing T_1 to satisfy the given (X, Y)-privacy on the join of T_1 and T_2. We can first apply Corollary 4.2 to test if this is possible, and below we assume it is. Let X_i denote $X \cap att(T_i)$, Y_i denote $Y \cap att(T_i)$, and J_i denote the join attributes in T_i, where $i = 1, 2$.

5.1 Overview

The algorithm, called *Top-Down Specialization for Sequential Anonymization (TDS4SA)*, is given in Algorithm 1. The input consists of T_1, T_2, the (X, Y)-privacy requirement, and the taxonomy tree for each categorical attribute in X_1. Starting from the most generalized T_1, the algorithm iteratively specializes the attributes A_j in X_1. T_1 contains the current set of *generalized records* and Cut_j contains the current set of *generalized values* for A_j. In each iteration, if some Cut_j contains a "valid" candidate for specialization, it chooses the winner w that maximizes $Score$. A candidate is *valid* if the join specialized by the candidate does not violate the privacy requirement. The algorithm then updates $Score(v)$ and status for the candidates v in $\cup Cut_j$. This process is repeated until there is no more valid candidate. On termination, Corollary 4.1 implies that a further specialization produces no solution, so T_1 is a maximally specialized state satisfying the given privacy requirement.

Below, we focus on the three key steps in Lines 3 to 5.

Algorithm 1 Top-Down Specialization for Sequential Anonymization

Input: T_1, T_2, a (X, Y)-privacy requirement, a taxonomy tree for each categorical attribute in X_1.

Output: a generalized T_1 satisfying the privacy requirement.

1: generalize every value of A_j to ANY_j where $A_j \in X_1$;
2: **while** there is a valid candidate in $\cup Cut_j$ **do**
3:    find the winner w of highest $Score(w)$ from $\cup Cut_j$;
4:    specialize w on T_1 and remove w from $\cup Cut_j$;
5:    update $Score(v)$ and the valid status for all v in $\cup Cut_j$;
6: **end while**
7: output the generalized T_1 and $\cup Cut_j$;

5.2 Score Metric

$Score(v)$ evaluates the "goodness" of a specialization v for preserving privacy and information. Each specialization gains some "information", $InfoGain(v)$, and loses some "privacy", $PrivLoss(v)$. We choose the specialization that maximizes the trade-off between the gain of information gain and the loss of privacy, proposed in [5]:

$$Score(v) = \frac{InfoGain(v)}{PrivLoss(v) + 1}. \quad (1)$$

$InfoGain(v)$ is measured on T_1 whereas $PrivLoss(v)$ is measured on the join of T_1 and T_2.

Consider a specialization $v \rightarrow \{v_1, \cdots, v_c\}$. For a continuous attribute, $c = 2$, and v_1 and v_2 represent the binary split of the interval v that maximizes $InfoGain(v)$. Before the specialization, $T_1[v]$ denotes the set of generalized records in T_1 that contain v. After the specialization, $T_1[v_i]$ denotes the set of records in T_1 that contain v_i, $1 \leq i \leq c$.

The choice of $InfoGain(v)$ and $PrivLoss(v)$ depends on the information requirement and privacy requirement. If T_1 is released for classification on a specified class column, $InfoGain(v)$ could be the reduction of the class entropy [15], defined by

$$InfoGain(v) = Ent(T_1[v]) - \sum_i \frac{|T_1[v_i]|}{|T_1[v]|} Ent(T_1[v_i]). \quad (2)$$

$Ent(R)$ is the class entropy of a set of records R following from Shannon's information theory [18]. The more dominating the majority class in R is, the smaller $Ent(R)$ is and the smaller the classification error is. The computation depends only on the class frequency and some count statistics of v and v_i in $T_1[v]$ and $T_1[v_1] \cup \cdots \cup T_1[v_c]$. Another choice of $InfoGain(v)$ could be the notion of distortion [17]. If generalizing a child value v_i to the parent value v costs one unit of distortion, the information gained by the specialization $v \rightarrow \{v_1, \cdots, v_c\}$ is

$$InfoGain(v) = |T_1[v]|. \quad (3)$$

The third choice can be the discernibility [2].

For (X, Y)-privacy, $PrivLoss(v)$ is measured by the decrease of $A_Y(X)$ or the increase of $L_Y(X)$ due to the specialization of v: $A_Y(X) - A_Y(X_v)$ for (X, Y)-anonymity, and $L_Y(X_v) - L_Y(X)$ for (X, Y)-linkability, where X and X_v represent the attributes before and after specializing v respectively. Computing $PrivLoss(v)$ involves the count statistics about X and Y over the join of T_1 and T_2, before and after the specialization of v, which can be expensive.

Challenges. Though Algorithm 1 has a simple high level structure, several computational challenges must be resolved for an efficient implementation. First, each specialization of the winner w

affects the matching of join, hence, the checking of the privacy requirement (i.e., the status on Line 5). It is extremely expensive to rejoin the two tables for each specialization performed. Second, it is inefficient to "perform" every candidate specialization v just to update $Score(v)$ on Line 5 (note that $A_Y(X_v)$ and $L_Y(X_v)$ are defined for the join assuming the specialization of v is performed). Moreover, materializing the join is impractical because a lossy join can be very large. A key contribution of this work is an efficient solution that incrementally maintains some count statistics without executing the join. We consider the two types of privacy separately.

5.3 (X,Y)-Linkability

Two expensive operations on performing the winner specialization w are accessing the records in T_1 containing w and matching the records in T_1 with the records in T_2. To support these operations efficiently, we organize the records in T_1 and T_2 into two tree structures. Recall that $X_1 = X \cap att(T_1)$ and $X_2 = X \cap att(T_2)$, and J_1 and J_2 denote the join attributes in T_1 and T_2.

Tree1 and Tree2. In *Tree1*, we partition the T_1 records by the attributes X_1 and $J_1 - X_1$ in that order, one level per attribute. Each root-to-leaf path represents a generalized record on $X_1 \cup J_1$, with the partition of the original records generalized being stored at the leaf node. For each generalized value v in Cut_j, $Link[v]$ links up all nodes for v at the attribute level of v. Therefore, $Link[v]$ provides a direct access to all T_1 partitions generalized to v. Tree1 is updated upon performing the winner specialization w in each iteration. In *Tree2*, we partition the T_2 records by the attributes J_2 and $X_2 - J_2$ in that order. No specialization is performed on T_2, so Tree2 is static. Some "count statistics", described below, are stored for each partition in Tree1 and Tree2.

Specialize w (Line 4). This step performs the winner specialization $w \to \{w_1, \cdots, w_c\}$, similar to the TDS algorithm for a single release in [5]. It follows $Link[w]$, and for each partition P_1 on the link,

- Step 1: refine P_1 into the specialized partitions for w_i, link them into $Link[w_i]$. The specialized partitions remain on the other links of P_1. This step will scan the raw records in P_1. In the *same* scan, we also collect the following *count statistics* for each (new) partition P on $Link[w_i]$, which will be used later to update $Score(v)$. Let $P[u]$ denote the subset of P containing the value u and $|P|$ denote the size of P:
  - $|P|, |P[\theta]|, |P[w_{ij}]|$ and $|P[w_{ij}, \theta]|$ (for Equation (2)).
  - $|P|$ (for Equation (3)).
  - $|P[y]|$ and $|P[w_{ij}, y]|$ if Y is in T_1, or $|P|$ and $|P[w_{ij}]|$ if Y is in T_2 (for updating $L_Y(X_v)$).

θ is a class label in the class column, y is a value on Y, and w_{ij} is a child value of w_i. These count statistics are stored together with the partition P.

- Step 2: probe the matching partitions in Tree2. Match the last $|J_1|$ attributes in P_1 with the first $|J_2|$ attributes in Tree2. For each matching node at the level $|J_2|$ in Tree2, scan all partitions P_2 below the node. If x is the value on X represented by the pair (P_1, P_2), increment $a(x)$ by $|P_1| \times |P_2|$, increment $a(x, y)$ by $|P_1[y]| \times |P_2|$ if Y is in T_1, or by $|P_1| \times |P_2[y]|$ if Y is in T_2, where y is a value on Y. We employ an "X-tree" to keep $a(x)$ and $a(x, y)$ for the values x on X. In the X-tree, the x values are partitioned by the attributes X, one level per attribute, and are represented by leaf nodes. $a(x)$ and $a(x, y)$ are kept at the leaf node for x. Note that $l_y(x) = a(x, y)/a(x)$ and $L_y(X) = max\{l_y(x)\}$ over all the leaf nodes x in the X-tree.

Remarks. This step (Line 4) is the only time that raw records are accessed in our algorithm.

Update Score(v) (Line 5). This step updates $Score(v)$ for the candidates v in $\cup Cut_j$ using the count statistics collected at the partitions in Tree1 and $a(x)$ and $a(x, y)$ in the X-tree. The idea is the same as [5], so we omit the details. An important point is that this operation does not scan raw records, therefore, is efficient. This step also updates the "valid" status: If $L_Y(X_v) \leq k$, mark v as "valid".

Analysis. (1) The records in T_1 and T_2 are stored only once in Tree1 and Tree2. For the static Tree2, once it is created, data records can be discarded. (2) On specializing the winner w, $Link[w]$ provides a direct access to the records involved in T_1 and Tree2 provides a direct access to the matching partitions in T_2. Since the matching is performed at the partition level, not the record level, it scales up with the size of tables. (3) The cost of each iteration has two parts. The first part involves scanning the affected partitions on $Link[w]$ for specializing w in Tree1 and maintaining the count statistics. This is the only operation that accesses records. The second part involves using the count statistics to update the score and status of candidates. (4) In the whole computation, each record in T_1 is accessed at most $|X \cap att(T_1)| \times h$ times because a record is accessed only if it is specialized on some attribute from $X \cap att(T_1)$, where h is the maximum height of the taxonomies for the attributes in $X \cap att(T_1)$.

5.4 (X,Y)-Anonymity

Like for (X, Y)-linkability, we use Tree1 and Tree2 to find the matching partitions (P_1, P_2), and performing the winner specialization and updating $Score(v)$ is similar to Section 5.3. But now, we use the X-tree to update $a_Y(x)$ for the values x on X, and there is one important difference in the update of $a_Y(x)$. Recall that $a_Y(x)$ is the number of *distinct* values y on Y associated with the value x. Since the same (x, y) value may be found in more than one matching (P_1, P_2) pair, we cannot simply sum up the count extracted from all pairs. Instead, we need to keep track of distinct Y values for each x value to update $a_Y(x)$. In general, this is a time-consuming operation, e.g., requiring sorting/hashing/scanning. Below, we identify several special but important cases in which $a_Y(x)$ can be updated efficiently.

Case 1: X **contains all join attributes.** In this case, $J_1 \subseteq X_1$ and $J_2 \subseteq X_2$, and the partitioning in Tree1 and Tree2 is based on X_1 and X_2. Hence, each x value is contributed by *exactly one* matching (P_1, P_2) pair and is inserted into the X-tree *only once*. Therefore, there is no duplicate Y value for each x value. The computation is as follows: for each matching (P_1, P_2) pair, compute $a_Y(x_1 x_2)$ by $a_{Y_1}(x_1) \times a_{Y_2}(x_2)$, where x_i's ($i = 1, 2$) are represented by P_i's, and $a_{Y_i}(x_i)$'s are stored with the partitions P_i for x_i in Tree$i$. $a_{Y_i}(x_i) = 1$ if $Y_i = \emptyset$.

$a_{Y_1}(x_1)$ and $a_{Y_2}(x_2)$ are computed as follows. At the root of Tree1, we sort all records in the partition according to Y_1 (skip this step if $Y_1 = \emptyset$). For the value x_1 represented by the root, $a_{Y_1}(x_1)$ is equal to the number of *distinct* Y_1 values in the sorted list. On performing the winner specialization w, as we follow $Link[w]$ in Tree1 to specialize each partition P_1 on the link, we create the sorted list of records for the specialized partitions of P_1, which allows to compute $a_{Y_1}(x_{11}), \cdots, a_{Y_1}(x_{1c})$ for the specialized values $x_{11}, \cdots, x_{1c}$. Note that these lists are *automatically sorted* because their "parent" list is sorted. For the static Tree2, we can collect $a_{Y_2}(x_2)$ at each leaf node representing a value x_2 on X_2 in an initialization and subsequently never need to modify it.

Case 2: Y_2 **is a key in** T_2. In this case, the matching pairs (P_1, P_2) for the same value x do not share any common Y values; therefore, there is no duplicate Y value for x. To see this, let $Pair_x$ be the set of all matching pairs (P_1, P_2) representing x. Since all

Table 2: Attributes for the *Adult* data set

Dept.	Attribute	Type	Numerical Range	
			# Leaves	# Levels
Taxation	Age (Ag)	Cont.	17 - 90	
	Capital-gain (Cg)	Cont.	0 - 99999	
	Capital-loss (Cl)	Cont.	0 - 4356	
	Education-num (En)	Cont.	1 - 16	
	Final-weight (Fw)	Cont.	13492 - 1490400	
	Hours-per-week (H)	Cont.	1 - 99	
	Education (E)	Cat.	16	5
	Occupation (O)	Cat.	14	3
	Work-class (W)	Cat.	8	5
Common	Martial-status (M)	Cat.	7	4
	Relationship (Re)	Cat.	6	3
	Sex (S)	Cat.	2	2
Immigra-tion	Native-country (Nc)	Cat.	40	5
	Race (Ra)	Cat.	5	3

P_1's in $Pair_x$ have the same X value (i.e., x), they must have different join values on J_1 (otherwise they should not be different partitions). This means that each P_2 occurs in at most one pair (P_1, P_2) in $Pair_x$. Since P_2's are disjoint and Y_2 is a key of T_2, the pairs (P_1, P_2) in $Pair_x$ involve disjoint sets of Y_2 values, therefore, disjoint sets of Y values. This property ensures that, for each matching (P_1, P_2), $a_Y(x)$ can be computed by $a_{Y_1}(x_1) \times a_{Y_2}(x_2)$, where $a_{Y_1}(x_1)$ and $a_{Y_2}(x_2)$ are stored with P_1 in Tree1 and P_2 in Tree2, as in Case 1. Note that $a_{Y_2}(x_2)$ is equal to $|P_2|$ because Y_2 is a key of T_2.

Case 3: Y_1 is a key of T_1 and $Y_2 = \emptyset$. In this case, each P_1 in Tree1 involves $|P_1|$ distinct Y_1 values and shares no common Y values with other partitions. To update the X-tree, for each P_1 and all pairs (P_1, P_2) representing the same value x on X, we set $a_Y(x)$ to $|P_1|$ *only once*. Note that $Y_2 = \emptyset$ is required; otherwise we have to check for duplicates of Y values.

Case 4: Y is a key of the join of T_1 and T_2. For example, if $Y = \{K_1, K_2\}$, where K_i is a key in T_i. In this case, $a_Y(x)$ is equal to the number of records containing x in the join. Since each pair (P_1, P_2) involves a disjoint set of records in the join, we increment $a_Y(x)$ by $|P_1| \times |P_2|$ for the value x represented by (P_1, P_2).

6. EMPIRICAL STUDY

All experiments were conducted on an Intel Pentium IV 2.4GHz PC with 1GB RAM. The data set is the publicly available *Adult* data set from [14], previously used in [2][5][6][8]. There were 30,162 and 15,060 records without missing values for the pre-split training set and testing set respectively. We combined them into one set for generalization. Table 2 describes the attributes (Cat. for categorical and Cont. for continuous) and the binary Class corresponding to income levels $\leq$50K or >50K. We adopted the taxonomy trees in [5]. The data is released to two users. Taxation Department (T_1) is interested in the first 12 attributes and the Class attribute. Immigration Department (T_2) is interested in the last 5 attributes. Both are interested in the 3 common attributes in the middle, M, Re, S. We created two versions of the data set (T_1, T_2), Set A and Set B.

Set A (categorical attributes only): This data set contains only categorical attributes. T_1 contains the Class attribute, the 3 categorical attributes for Taxation Department and the 3 common attributes. T_2 contains the 2 categorical attributes for Immigration Department and the 3 common attributes. The top 6 ranked attributes in T_1 are M, Re, S, E, O, W in that order, ranked by

discriminative power on the Class attribute. The join attributes are the common attributes M, Re, S. The rationale is that if join attributes are not important, they should be removed first.

Set B (categorical and continuous attributes): In addition to the categorical attributes as in Set A, T_1 contains the additional 6 continuous attributes from Taxation Department. T_2 is the same as in Set A. The top 7 attributes in T_1 are Cg, Ag, M, En, Re, H, S in that order.

We consider two cost metrics. The "classification metric" is the classification error on the generalized testing set of T_1 where the classifier for Class is built from the generalized training set of T_1. The "distortion metric" was proposed in [17]. Each time a categorical value is generalized to the parent value in a record in T_1, there is one unit of distortion. For a continuous attribute, if a value v is generalized to an interval $[a, b]$, there is $(b - a)/(f_2 - f_1)$ unit of distortion for a record containing v, where $[f_1, f_2]$ is the full range of the continuous attribute. The distortion is separately computed for categorical attributes and continuous attributes. The total distortion is normalized by the number of records.

6.1 Results for (X,Y)-Anonymity

We choose X so that (1) X contains the N top ranked attributes in T_1 for a specified N (to ensure that the generalization is performed on important attributes), (2) X contains all join attributes (thus Case 1 in Section 5.4), and (3) X contains all attributes in T_2. TopN refers to the (X, Y)-anonymity so chosen. Below, K_i is a key in T_i, $i = 1, 2$. We compare the following error or distortion:

- XYE: the error produced by our method with $Y = K_1$.
- $XYE(row)$: the error produced by our method with $Y = \{K_1, K_2\}$.
- BLE: the error produced by the unmodified data.
- KAE: the error produced by k-anonymity on T_1 with $QID = att(T_1)$.
- RJE: the error produced by removing all join attributes from T_1.
- XYD: the distortion produced by our method with $Y = K_1$.
- KAD: the distortion produced by k-anonymity on T_1 with $QID = att(T_1)$.

The "benefit" and "loss" refer to the error/distortion reduction and increase by our method in comparison with another method.

Results for Set A. Figure 1 depicts KAD and XYD averaged over the thresholds $k = 40, 80, 120, 160, 200$, with $KAD - XYD$ being the benefit compared to k-anonymization. For Top3 to Top6, this benefit ranges from 1 to 7.16, which is significant considering $KAD = 9.23$. Figure 2 depicts the classification error averaged over the thresholds $k = 40, 80, 120, 160, 200$. $BLE = 17.5\%, RJE = 22.3\%, KAE = 18.4\%$. The main results are summarized as follows.

$XYE - BLE$: this is the loss of our method compared to the unmodified data. In all the cases tested, $XYE - BLE$ is at most 0.9%, with the error on the unmodified data being $BLE = 17.5\%$. This small error increase, for a wide range of privacy requirements, suggests that the information utility is preserved while anonymizing the database in the presence of previous releases.

$XYE - XYE(row)$: this is the loss due to providing anonymization wrt $Y = \{K_1\}$ compared to anonymization wrt $Y = \{K_1, K_2\}$. For the same threshold k, since $a_{K_1}(x) \leq a_{K_1, K_2}(x)$, the former requires more generalization than the latter. However, this experiment shows that the loss is no more than 0.2%. On the other hand, the anonymization with $Y = \{K_1, K_2\}$ failed to provide the anonymity wrt K_1. For example, for Top6 and $k = 200$, 5.5% of the X values linked to more than 200 values on $\{K_1, K_2\}$ were ac-
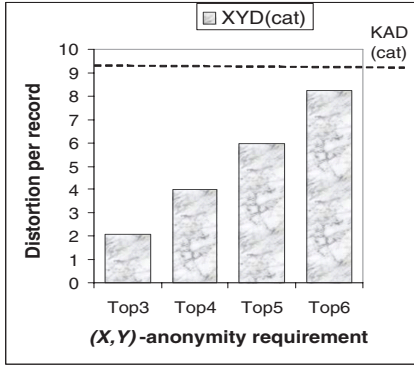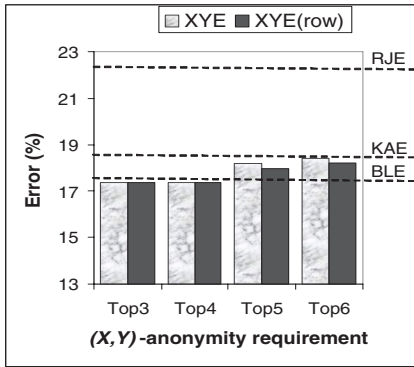
Figure 1: Distortion for (X, Y)-anonymity, Set A

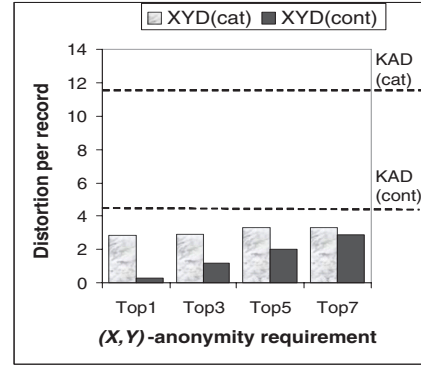**Figure 3: Distortions for (X, Y)-anonymity, Set B**

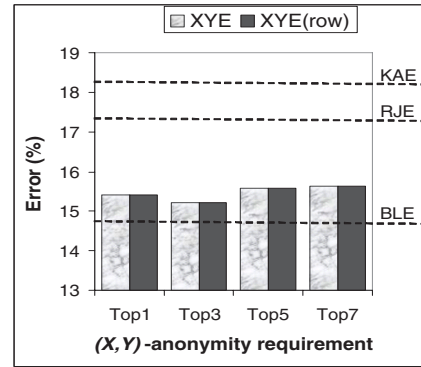**Figure 2: Errors for (X, Y)-anonymity, Set A**

Figure 4: Errors for (X, Y)-anonymity, Set B

tually linked to less than 200 distinct values on K_1. This problem cannot be easily addressed by a larger threshold k on the number of values for $\{K_1, K_2\}$ because the number of K_1 values involved can be arbitrarily low.

$RJE - XYE$: this is the benefit over the removal of join attributes. It ranges from 3.9% to 4.9%, which is significant considering the base line error $BLE = 17.5\%$. The benefit could be more significant if there are more join attributes. Since the attacker typically uses as many attributes as possible for join, simply removing join attributes is not a good solution.

$KAE - XYE$: this is the benefit over the k-anonymization on T_1. For Set A, this benefit is not very significant. The reason is that T_1 contains only 6 attributes, many of which are included in X to ensure that the generalization is not on trivial attributes. As a result, the privacy requirement becomes somehow similar to the standard k-anonymization on all attributes in T_1. However, Set B where T_1 contains more attributes, a more significant benefit was demonstrated.

Results for Set B. Figure 3 shows the distortion reduction compared to the k-anonymization of T_1, $KAD(cat) - XYD(cat)$ for categorical attributes, and $KAD(cont) - XYD(cont)$ for continuous attributes. For both types of attributes, the reduction is very significant. This strongly supports that the lossy join achieves privacy with less data distortion. Figure 4 depicts the classification error. $BLE = 14.7\%$, $RJE = 17.3\%$, and averaged $KAE = 18.2\%$. The main results are summarized as follows.

$XYE - BLE$: this loss is averaged at 0.75%, a slight increase of error compared to the unmodified data.

$XYE - XYE(row)$: We observed no loss for achieving the more restrictive anonymization wrt $Y = \{K_1\}$ compared to wrt $Y = \{K_1, K_2\}$. We noted that both methods bias toward continuous attributes and all join (categorical) attributes are fully general-

ized to the top value ANY. In this case, every record in T_1 matches every record in T_2, which makes $a_{K_1}(x)$ and $a_{K_1,K_2}(x)$ equal.

$RJE - XYE$: this benefit is smaller than in Set A. For Set B, join attributes are less critical due to the inclusion of continuous attributes, and the removal of join attributes results in a more gentle loss.

$KAE - XYE$: this benefit is more significant than in Set A. The k-anonymization of T_1 suffers from a more drastic generalization on QID that now contains both continuous and categorical attributes in T_1. As a result, our benefit of not generalizing all attributes in T_1 is more evident in this data set.
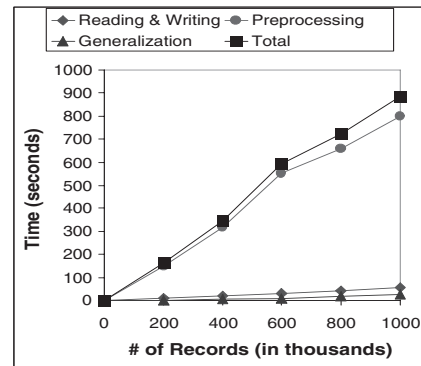

Figure 5: Scalability for (X, Y)-anonymity ($k = 40$)

Scalability. For all the above experiments, our algorithm took less than 30 seconds, including disk I/O operations. To further evaluate its scalability, we enlarged Set A as follows. Originally, both T_1 and T_2 contain 45,222 records. For each original record r in a

table T_i, we created $\alpha - 1$ "variations" of r, where $\alpha > 1$ is the expansion scale. For each variation of r in T_i, we assigned a unique identifier for K_i, randomly and uniformly selected q attributes from X_i, $i = 1, 2$, randomly selected some values for these q attributes, and inherited the other values from the original r. The rationale of variations is to increase the number of partitions in Tree1 and Tree2. The enlarged data set has $\alpha \times 45,222$ records in each table. We employed the Top6 (X, Y)-anonymity requirement with $Y = K_1$ and $k = 40$ in Set A. Other choices require less runtime.

Figure 5 depicts the runtime distribution in different phases of our method for 200K to 1M data records in each table. Our method spent 885 seconds in total to transform 1M records in T_1. Approximately 80% of the time was spent on the preprocessing phase, i.e., sorting records in T_1 by K_1 and building Tree2. Generalizing T_1 to satisfy the (X, Y)-anonymity took less than 4% of the total time.

6.2 Results for (X,Y)-Linkability

In this experiment, we focused on the classification error because the distortion due to (X, Y)-Linkability is not comparable with the distortion due to k-anonymity. For Set A, we specified four (X, Y)-linkability requirements, denoted Top1, Top2, Top3 and Top4, such that Y contains the top 1, 2, 3 and 4 categorical attributes in T_1. The rationale is simple: if Y does not contain important attributes, removing all attributes in Y from T_1 would provide an immediate solution. We specified the 50% least frequent (therefore, most vulnerable) values of each attribute in Y as the sensitive properties y. X contains all the attributes in T_1 not in Y, except $T_2.Ra$ and $T_2.Nc$ because otherwise no privacy requirement can be satisfied. For Set B, T_1 and X contain the 6 continuous attributes, in addition to the categorical attributes in Set A. Besides XYE, BLE and RJE in Section 6.1, RSE denotes the error produced by removing all attributes in Y from T_1.

Results for Set A. Figure 6 shows the averaged error over thresholds $k = 10\%, 30\%, 50\%, 70\%, 90\%$. $BLE = 17.5\%$ and $RJE = 22.3\%$. $XYE - BLE$ is no more than 0.7%, a small loss for a wide range of (X, Y)-linkability requirement compared to the unmodified data. $RSE - XYE$ is the benefit of our method over the removal of Y from T_1. It varies from -0.2% to 5.6% and increases as more attributes are included in Y. $RJE - XYE$ spans from 4.1% to 4.5%, showing that our method better preserves information than the removal of join attributes.

Results for Set B. Figure 7 depicts the averaged XYE and RSE. $BLE = 14.7\%$ and $RJE = 17.3\%$. XYE is 15.8%, 1.1% above BLE. $RSE - XYE$ spans from 0.1% to 1.9%, and $RJE - XYE$ spans from 0.7% to 2.6%. These benefits are smaller than in Set A because continuous attributes in Set B took away classification from categorical attributes. In other words, the removal of join attributes or attributes in Y, all being categorical attributes, causes less error. However, XYE consistently stayed below RSE and RJE.

Scalability. Our algorithm took less than 20 seconds in Set A and less than 450 seconds in Set B, including disk I/O operations. The longest time was spent on Set B for (X, Y)-linkability because the interval for a continuous attribute is typically split many times before the maximum linkability is violated. For scalability evaluation, we used the Top1 requirement described above for Set A and $k = 90\%$. We enlarged Set A as described in Section 6.1, but the values for Y are inherited from the original r instead of being assigned unique identifiers. Figure 8 depicts the runtime distribution of our method with 200K to 1M data records in each table. Our method spent 83 seconds to transform 1M records in T_1. The preprocessing phase, i.e., building Tree2, took less than 1 second. Generalizing T_1 to satisfy the (X, Y)-linkability took 25 seconds.
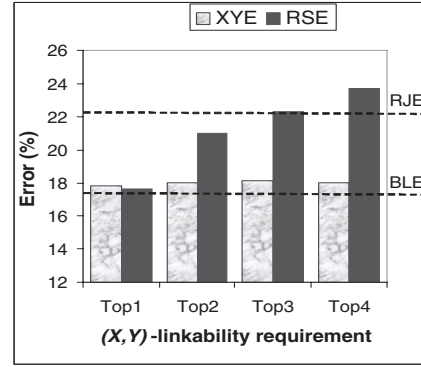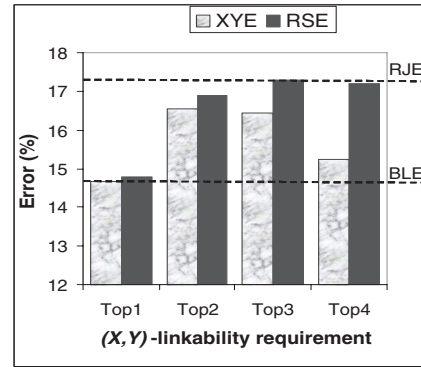

Figure 6: Errors for (X, Y)-linkability, Set A

Figure 7: Errors for (X, Y)-linkability, SetB

6.3 Summary

The proposed method pays a small data penalty to achieve a wide range of (X, Y)-privacy in the scenario of sequential releases. The method is superior to several obvious candidates, k-anonymization, removal of join attributes, and removal of sensitive attributes, which do not respond dynamically to the (X, Y)-privacy specification and the generalization of join. The experiments showed that the dynamical response to the generalization of join helps achieve the specified privacy with less data distortion. The proposed index structure is highly scalable for anonymizing large data sets.

7. EXTENSIONS

We now extend this approach to the general case that more than one previously released tables $T_2, \cdots, T_p$. One solution is first joining all previous releases $T_2, \cdots, T_p$ into one "history table" and then applying the proposed method for two releases. This history table is likely extremely large because all $T_2, \cdots, T_p$ are some generalized versions and there may be no join attributes between them. A preferred solution should deal with all releases at their original size. Our insight is that, as remarked at the end of Section 4, Lemma 4.1-4.2 can be extended to this general case. Below, we extend some definitions and modification required for the top-down specialization in Section 5.

Let t_i be a record in T_i. The *Consistency Predicate* states that, for all releases T_i that have a common attribute A, $t_i.A$'s are on the same generalization path in the taxonomy tree for A. The *Inconsistency Predicate* states that for distinct attributes $T_i.A$ and $T_j.B$, $t_i.A$ and $t_j.A$ are not semantically inconsistent according the "common sense". $(t_1, t_2, \cdots, t_p)$ is a *match* if it satisfies both predicates. The *join* of $T_1, T_2, \cdots, T_p$ is a table that contains all matches $(t_1, t_2, \cdots, t_p)$. For a (X, Y)-privacy on the join, X and
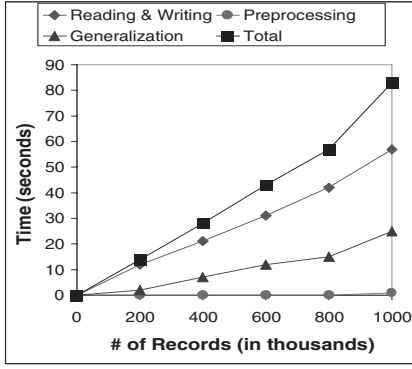
Figure 8: Scalability for (X,Y)-linkability ($k=90\%$)

Y are disjoint subsets of $att(T_1) \cup att(T_2) \cup \cdots \cup att(T_p)$ and if X contains a common attribute A, X contains all $T_i.A$ such that T_i contains A.

DEFINITION 7.1 (SEQUENTIAL ANONYMIZATION). Suppose that tables $T_2, \cdots, T_p$ were previously released. The data holder wants to release a table T_1, but wants to ensure a (X,Y)-privacy on the join of $T_1, T_2, \cdots, T_p$. The *sequential anonymization* is to generalize T_1 on the attributes in $X \cap att(T_1)$ such that the join satisfies the privacy requirement and T_1 remains as useful as possible. ∎

We consider only (X,Y)-linkability for the top-down specialization; the extension for (X,Y)-anonymity can be similarly considered. For simplicity, we assume that previous releases $T_2, \cdots, T_p$ have a *star join* with T_1: every T_i ($i > 1$) joins with T_1. On performing the winner specialization w, we use Tree$i$, $i = 1, \cdots, p$, to probe matching partitions in T_i. Let $J_i(j)$ denote the set of join attributes in T_i with T_j. Let $X_i = X \cap att(T_i)$ and $Y_i = Y \cap att(T_i)$. Tree1 is partitioned by $X_1 \cup J_1(2) \cup \cdots \cup J_1(p))$. For $i = 2, \cdots, p$, Tree$i$ is partitioned by $J_i(1)$ and $X_i - J_i(1)$. As in Section 5, we identify the partitions on $Link[w]$ in Tree1. For each partition P_1 on the link, we probe the matching partitions P_i in Tree$i$ by matching $J_i(1)$ and $J_1(i)$, $1 < i \le p$. Let $(P_1, \cdots, P_p)$ be such that P_1 matches P_i, $2 \le i \le p$. If $(P_1, \cdots, P_p)$ satisfies both predicates, we update the X-tree for the value x represented by $(P_1, \cdots, P_p)$: increment $a(x,y)$ by $s_1 \times \cdots \times s_p$ and increment $a(x)$ by $|P_1| \times \cdots \times |P_p|$, where $s_i = |P_i|$ if $Y_i = \emptyset$, and $s_i = |P_i[y_i]|$ if $Y_i \ne \emptyset$.

8. CONCLUSION

Previous works on k-anonymization focused on a single release of data. In reality, data is not released in one-shot, but released continuously to serve various information purposes. The availability of related releases enables sharper identification attacks through a global quasi-identifier made up of the attributes across releases. In this paper, we studied the anonymization problem of the current release under this assumption, called *sequential anonymization*. We extended the privacy notion to this case. We introduced the notion of lossy join as a way to hide the join relationship among releases. We addressed several computational challenges raised by the dynamic response to the generalization of join, and we presented a scalable solution to the sequential anonymization problem.

9. REFERENCES

[1] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *ICDT*, 2005.

[2] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *IEEE ICDE*, pages 217–228, 2005.

[3] C. Clifton. Using sample size to limit exposure to data mining. *Journal of Computer Security*, 8(4):281–307, 2000.

[4] A. Deutsch and Y. Papakonstantinou. Privacy in database publishing. In *ICDT*, 2005.

[5] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *IEEE ICDE*, pages 205–216, April 2005.

[6] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *ACM SIGKDD*, pages 279–288, 2002.

[7] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In *ACM SIGMOD*, Chicago, IL, June 2006.

[8] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *ACM SIGMOD*, 2005.

[9] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *IEEE ICDE*, 2006.

[10] A. Machanavajjhala, J. Gehrke, and D. Kifer. l-diversity: Privacy beyond k-anonymity. In *IEEE ICDE*, 2006.

[11] B. Malin and L. Sweeney. How to protect genomic data privacy in a distributed network. In *Journal of Biomed Info, 37(3): 179-192*, 2004.

[12] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, 2004.

[13] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *ACM SIGMOD*, 2004.

[14] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[16] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[17] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In *IEEE Symposium on Research in Security and Privacy*, May 1998.

[18] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379 and 623, 1948.

[19] L. Sweeney. k-Anonymity: a model for protecting privacy. In *International Journal on Uncertanty, Fuzziness and Knowledge-based Systems, 10(5)*, pages 557–570, 2002.

[20] K. Wang, B. C. M. Fung, and G. Dong. Integrating private databases for data analysis. In *IEEE ISI*, May 2005.

[21] K. Wang, B. C. M. Fung, and P. S. Yu. Template-based privacy preservation in classification problems. In *IEEE ICDM*, pages 466–473, November 2005.

[22] K. Wang, B. C. M. Fung, and P. S. Yu. Handicapping attacker's confidence: An alternative to k-anonymization. *Knowledge and Information Systems: An International Journal*, 2006.

[23] K. Wang, P. S. Yu, and S. Chakraborty. Bottom-up generalization: A data mining solution to privacy protection. In *IEEE ICDM*, November 2004.

[24] R. C. W. Wong, J. Li., A. W. C. Fu, and K. Wang. (α,k)-anonymity: An enhanced k-anonymity model for privacy preserving data publishing. In *ACM SIGKDD*, 2006.

[25] X. Xiao and Y. Tao. Personalized privacy preservation. In *ACM SIGMOD*, June 2006.

[26] C. Yao, X. S. Wang, and S. Jajodia. Checking for k-anonymity violation by views. In *VLDB*, 2005.