k-Anonymity-Based Horizontal Fragmentation to Preserve Privacy in Data Outsourcing

Abbas Taheri Soodejani, Mohammad Ali Hadavi, and Rasool Jalili

Data and Network Security Laboratory,
Department of Computer Engineering, Sharif University of Technology
{a.taheri@cert.,mhadavi@ce.,jalili@}sharif.edu

Abstract. This paper proposes a horizontal fragmentation method to preserve privacy in data outsourcing. The basic idea is to identify sensitive tuples, anonymize them based on a privacy model and store them at the external server. The remaining non-sensitive tuples are also stored at the server side. While our method departs from using encryption, it outsources all the data to the server; the two important goals that existing methods are unable to achieve simultaneously. The main application of the method is for scenarios where encrypting or not outsourcing sensitive data may not guarantee the privacy.

Keywords: Data outsourcing, privacy, horizontal fragmentation, *k*-anonymity.

1 Introduction

In fragmentation-based approach, some data columns are separated from each other to hide their sensitive associations, called vertical fragmentation; and also some sensitive tuples are separated from non-sensitive tuples, called horizontal fragmentation.

From the owner involvement in data storage view, fragmentation-based methods fall into two categories: (1) **Partial-outsourcing** methods [1-3], which store a portion of data at the owner side, (2) **Full-outsourcing** methods [4-6], which completely outsource the data to the external server. Partial-outsourcing methods get involved the owner in data storage and consequently data management, which is largely in contradiction with the goal of outsourcing, i.e., outsourcing data management. On the other hand, full-outsourcing methods outsource all data exploiting data encryption. Consequently, they suffer from the same disadvantages as the encryption-based approach.

Our method is based on horizontal fragmentation. The main idea is to identify and *k*-anonymize the sensitive tuples. Anonymized tuples are stored as a fragment and non-sensitive tuples are stored as a logically separate fragment, both at the server side.

This paper proposes a full-outsourcing method that unlike the existing full-outsourcing methods does not use encryption but anonymization to provide privacy. The method is appropriate for scenarios that even encrypting or even not outsourcing the sensitive data cannot guarantee the privacy. In addition, our method inherits some benefits of the horizontal fragmentation in [2] such as is consistency with database normalization techniques, content-aware fragmentation, introducing a logical formalism for our fragmentation and controlling inference using data dependencies.

N. Cuppens-Boulahia et al. (Eds.): DBSec 2012, LNCS 7371, pp. 263–273, 2012.

The rest of this paper is organized as follows. Section 2 introduces the basic concepts and definitions. Section 3 presents an algorithm for our fragmentation method. Finally, section 4 concludes the paper.

2 Basic Concepts

Our view of relational databases is the formalism of first-order predicate logic with equality. We describe how to formalize some relational concepts with this formalism.

Relational instance: we view an instance I of a relational database schema $\mathcal{R}$ as a set of expressions of the form $R(a_1, a_2, \ldots, a_n)$, where R is an *n-ary* relation name in $\mathcal{R}$ and the a_i's are constants. Such expressions are called *ground tuples*.

Illness

Name	Disease
Andy	Hypertension
Alice	Obesity
Bob	Aids
David	Heart disease
Linda	Cholera
Sara	Flu
Tom	Viral disease

Treatment

Name	Medicine
Andy	Med A
Alice	Med B
David	Med C
Bob	Med D
Bob	Med E
Tom	Med X
Tom	Med Y

Info

Name	DoB	ZIP
Andy	1981/01/03	94142
Alice	1953/10/07	86342
Bob	1952/02/12	79232
David	1999/01/20	20688
Linda	1989/01/03	94139
Sara	2000/01/20	40496
Tom	1970/11/01	23567

Fig. 1. An instance (I) of a database schema

EXAMPLE 1. Consider the database instance I in Figure 1 (our running example throughout the paper). The schema is $\mathcal{R}$ = {*Illness*, *Treatment*, *Info*}, consisting of three relations. The instance I of $\mathcal{R}$ is a set of ground tuples *Illness*(Andy, Hypertension), *Treatment*(Andy, Med A), *Info*(Andy, 1981/01/03 , 94142), and the rest. □

Database Dependency: Database dependencies are domain-specific declarations reflecting the intended meaning of the stored data in a database. Along with the definition of a schema $\mathcal{R}$, a set of data dependencies $\mathcal{D}$ is defined that consists of tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs).

Definition 1 (Tuple-generating dependency). *A* tuple-generating dependency (TGD) *is a closed formula of the form* $\forall \boldsymbol{x}\ (\phi(\boldsymbol{x}) \longrightarrow \exists \boldsymbol{y}\ \psi(\boldsymbol{x}, \boldsymbol{y}))$, *where* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are vectors of variables;* $\phi(\boldsymbol{x})$ *is a (possibly empty) conjunction of atomic formulas, all with variables among the variables in* $\boldsymbol{x}$*;* $\psi(\boldsymbol{x}, \boldsymbol{y})$ *is a conjunction of atomic formulas, all with variables among the variables in* $\boldsymbol{x}$ *and* $\boldsymbol{y}$.

Definition 2 (Equality-generating dependency). *An* equality-generating dependency (EGD) *is a closed formula of the form* $\forall \boldsymbol{x}\ (\phi(\boldsymbol{x}) \longrightarrow \psi(\boldsymbol{x}))$, *where* $\boldsymbol{x}$ *is a vector of variables;* $\phi(\boldsymbol{x})$ *is a conjunction of atomic formulas, all with variables among the variables in* $\boldsymbol{x}$*;* $\psi(\boldsymbol{x})$ *is a conjunction of formulas of the form* $x = x'$, *where* x *and* x' *are distinct variables in* $\boldsymbol{x}$.

For a dependency, we call ϕ the *body* and ψ the *head* of dependency, respectively.

EXAMPLE 2. In our example, set of dependencies $\mathcal{D}$ can contain the following formulas:

d_1: $\forall$n, d (*Illness*(n, d) $\longrightarrow$ $\exists$b, z *Info*(n, b, z))
d_2: $\forall$n, m (*Treatment*(n, m) $\longrightarrow$ $\exists$d *Illness*(n, d))
d_3: $\forall$n (*Treatment*(n, Med D) $\wedge$ *Treatment*(n, Med E) $\longrightarrow$ *Illness*(n, Aids))
d_4: $\forall$n, d, z, d′, z′ (*Info*(n, d, z) $\wedge$ *Info*(n, d′, z′) $\longrightarrow$ (d = d′ $\wedge$ z = z′))

The TGD d_1 states that if a person is ill, his/her personal information must be available; that is, for each tuple for a person, say 'n', in the relation *Illness*, there must be a tuple for him/her in the relation *Info*. The TGD d_2 states that if a person takes a medicine, there must be a disease s/he contracted. The TGD d_3 states that if someone takes two medicines 'Med D' and 'Med E', s/he is certainly contracted disease 'Aids'. The EGD d_4, states that personal information of each person is unique. □

For a database schema $\mathcal{R}$, a set of dependencies $\mathcal{D}$, and an instance I of $\mathcal{R}$, there must not exist a dependency $d \in \mathcal{D}$ that is violated by I. The meaning of dependency violation is captured by Definition 3.

Definition 3 (Dependency violation). *Let* I *be a database instance. The dependency* d *is said to be violated by* I *if:*

- *There exists a vector of constants* $\boldsymbol{a}$ *such that the instantiation of the body* $\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ *of variables* $\boldsymbol{x}$ *with constants* $\boldsymbol{a}$ *holds in* I: $I \vDash \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$
- *but the instantiated head* $\exists \boldsymbol{y}\ \psi(\boldsymbol{x}, \boldsymbol{y})$ *of variables* $\boldsymbol{x}$ *with constants* $\boldsymbol{a}$ *is false in* I: $I \nvDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}]$ *if* d *is a TGD (similarly,* $\psi(\boldsymbol{x})$ *is false in* I: $I \nvDash \psi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ *if* d *is an EGD*).

In other words, a dependency d is said to be violated if there exists a set of ground tuples in I from which the body of d can be instantiated but no tuples exist in I that the head of d can be fully instantiated from.

If the instance I violates some dependencies in $\mathcal{D}$, a procedure called *Chase* [7, 8] is run on I. Running the chase on I, fixes the violated dependencies. In other words, the result of the chase is an instance, called *chased instance*, that satisfies all dependencies in $\mathcal{D}$. We refer the reader to [2, 9] for more details about the chase procedure.

2.1 Syntax of Privacy Constraints

Data owner's privacy requirements are modeled through privacy constraints:

Definition 4 (Privacy constraint). *Given a database schema* $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$, *a* privacy constraint c *is a closed formula of the form* $\exists \boldsymbol{x}\ \alpha(\boldsymbol{x})$, *where* $\boldsymbol{x}$ *is a vector of variables;* $\alpha(\boldsymbol{x})$ *is a conjunction of positive atomic formulas, all with variables among the variables in* $\boldsymbol{x}$ *and constants from the domains of attributes.*

In the above definition, $\alpha(\boldsymbol{x})$ is a conjunction of partial instantiation of R_i's in $\mathcal{R}$. Definition 4 states that if there exist tuples in the database instance that yield an

instantiation of the formula $\alpha(\boldsymbol{x})$, then there is a privacy violation, we say *a sensitive knowledge is disclosed*. The set of tuples that violate a privacy constraint is called the *violation set* denoted by *V*. Note that we restrict the syntax of privacy constraints to formulas without negation and with only conjunction as logical connective.

EXAMPLE 3. For the database schema $\mathcal{R}$ in our example, the set of privacy constraints, denoted by $\mathcal{C}$, may contain the following formulas:

c_1: ∃n *Illness*(n, Aids)
c_2: ∃d *Illness*(Sara, d)
c_3: ∃n, n′ (*Illness*(n, Cholera) *Treatment*(n′, Med X) *Treatment* (n′, Med Y))

where c_1 states that if there exists a tuple with value 'Aids' for attribute *Disease*, a privacy violation occur. In this case, the name of the person who contracted Aids is the sensitive knowledge. Similar interpretation holds for c_2. The constraint c_3 states that if there exist three tuples, one from relation *Illness* with name 'n' and disease 'Cholera' and the other two from relation *Treatment* with the same name 'n′' but one with medicine 'Med X' and the other with medicine 'Med Y', then these tuples together violate the privacy. Here, the sensitive knowledge is a sensitive association between *n* and *n′*, e.g., *n′* will be contracted Cholera (say, because Cholera is an infectious disease and all persons in the database live in the same place). □

2.2 *k*-Anonymity

In some scenarios, encrypting or not outsourcing sensitive data may not guarantee the privacy. For example, consider a healthcare database of patients' records. Let the records of the patients that contracted Aids be sensitive. To provide privacy, one may encrypt or not outsource the records of the patients with disease Aids. Alice has disease Aids, hence she has a record in the database. The attacker Bob knows that Alice has a disease but not exactly which kind of disease. He also knows that records with disease Aids are sensitive. He examines the data and observes that there is no record for Alice in the database. Therefore, he can infer that Alice has disease Aids. There are other scenarios that encrypting or suppressing data do not guarantee the privacy. In these scenarios, we exploit the *k*-anonymity concept to provide privacy.

Our method, inspired from the *k*-anonymity concept [10], aims to provide privacy of degree *k*. To this aim, some fake tuples is produced in a way that for each sensitive knowledge at least *k* - 1 sensitive but fake knowledge can be inferred from database instance. Thus, the probability of a sensitive knowledge being real is equal to or less than 1/*k*.

Let *c* be a violated constraint and *V* be its violation set. To provide privacy with degree *k* for *c*, we produce *k* - 1 (possibly overlapping) violation sets, all violating the same constraint *c*. Thus, for *c*, we have *k* violation sets and the sensitive knowledge of *c* is anonymized with *k* - 1 fake knowledge.

For the sake of simplicity, for each violated constraint we assume that the number of sensitive tuples in its violation set is one (generalizing the concepts to violation sets with more than one tuple is straightforward). Also, we assume that the sensitive

knowledge is the value of an attribute *S*, called *sensitive attribute*. With this assumption, the *k*-anonymity concept in our method is captured by the following principle:

Definition 5 (*k*-anonymity principle). *For a violation set of a constraint* c *with sensitive attribute* S*, there must be at least* k - 1 *fake violation sets all instantiated from* c*, and the set of* k *violation sets contains at least* k *distinct values for attribute* S.

EXAMPLE 4. Two tuples *Illness*(Bob, Aids) and *Illness*(Sara, Flu) violate the constraints c_1 and c_2, respectively, from Example 3. Let the attribute *Name* for c_1 and *Disease* for c_2 be the sensitive attributes. We produce tuples *Illness*(Jim, Aids) and *Illness*(Sara, Influenza) to 2-anonymize these two violation sets, respectively. □

2.3 Syntax of Anonymization Rules

As mentioned in the previous section, the sensitive knowledge inferable from those tuples violating a privacy constraint should be anonymized. For each privacy constraint we define an anonymization rule that states which attribute or combination of attributes (that are sensitive) in which tuples should be anonymized.

Definition 6 (Anonymization rule). *An* anonymization rule *for a privacy constraint* c *of the form* $\exists \boldsymbol{x}\ \alpha(\boldsymbol{x})$*, is a formula of the form* $\exists \boldsymbol{x}\ \alpha(\boldsymbol{x}) \xrightarrow{k-1} \exists \boldsymbol{y}\ \beta(\boldsymbol{y})$*, where* $\boldsymbol{x}$ *and* $\alpha(\boldsymbol{x})$ *are those defined in Definition 4;* $\boldsymbol{y}$ *is a (possibly empty) vector of variables;* $\beta(\boldsymbol{y})$ *is a conjunction of positive formulas all with free variables, bounded variables, and constants; and* k *is the k-anonymity parameter.*

In Definition 6, $\alpha(\boldsymbol{x})$ and $\beta(\boldsymbol{y})$ are called the *body* and *head* of the rule, respectively. According to the head we determine which tuples and, more specifically, which attributes should be anonymized. The above formula states that if there exist tuples that instantiate the body $\alpha(\boldsymbol{x})$, consequently violating a constraint, then there must exist *k* - 1 fake instantiations for the head $\beta(\boldsymbol{y})$. In fact, for a violated constraint, we apply an anonymization rule to achieve the *k*-anonymity principle in Definition 5. In $\beta(\boldsymbol{y})$, constants are values of the attributes, such as name, that their values can identify a sensitive knowledge; free variables represent a set of attributes, such as disease, that their values are sensitive; and bounded variables represent the attributes that have no role in the identification and have no association with the sensitive knowledge.

EXAMPLE 5. The set of anonymization rules, denoted by $\mathcal{A}$, corresponding to the privacy constraints in Example 3 contains the following rules:

r_1: $\exists$n *Illness*(n, Aids) $\xrightarrow{k-1}$ $Illness_{FS}$(n, Aids)

r_2: $\exists$d *Illness*(Sara, d) $\xrightarrow{k-1}$ $Illness_{FS}$(Sara, d)

r_3: $\exists$n, n′ (*Illness*(n, Cholera) *Treatment*(n′, Med X) *Treatment*(n′, Med Y)) $\xrightarrow{k-1}$ ($Treatment_{FS}$(n′, Med X) $Treatment_{FS}$(n′, Med Y))

Relation names above with subscripted *FS* indicate relations in the fragment F_S. The rule r_1 states that to anonymize a tuple that instantiates *Illness*(n, Aids), violating

c_1, k - 1 fake tuples should be produced that are instantiations of $Illness_{FS}$(n, Aids). In the set of k - 1 fake tuples, the attribute *Disease* should take the value 'Aids' and the attribute *Name* should be anonymized by taking k - 1 distinct values. Similar interpretation holds for r_2. The rule r_3 states that if there exist tuples that instantiate c_3, the sensitive knowledge can be anonymized by generating 2(k - 1) fake tuples; k - 1 tuples that instantiate $Treatment_{FS}$(n′, Med X), and k - 1 tuples that instantiate $Treatment_{FS}$(n′, Med Y). For the attribute *Name* in the latter k - 1 tuples, we use the same values used in the former k - 1 tuples. □

3 Fragmentation

In this section, we first define the requirements of a correct horizontal fragmentation. Then, we introduce an algorithm to produce a correct fragmentation.

3.1 Fragmentation Correctness

A fragmentation is correct if it satisfies three requirements: **completeness**, **non-redundancy**, and **privacy**. The completeness requirement states that we must be able to reconstruct the original database instance from its corresponding fragments. The non-redundancy requirement states that the fragments should not have common tuples and/or attributes (depending on the fragmentation method). The privacy requirement states that the fragmentation must satisfy the privacy constraints so that no sensitive knowledge can be inferred from the fragments. The completeness and privacy are two mandatory requirements that any fragmentation method must satisfy them, while the non-redundancy requirement can be considered optional as it is not applicable to all methods. We define fragmentation correctness in our method as follows:

Definition 7 (Fragmentation correctness). *Let $\mathcal{R}$ be a database schema with a set of dependencies $\mathcal{D}$. Let $\mathcal{C}$ be a set of privacy constraints and* k *be the k-anonymity parameter. Also let $\mathcal{F} = \{F_{NS}, F_S\}$ be a fragmentation for instance* I *of $\mathcal{R}$, where* F_S *and* F_{NS} *are sensitive and non-sensitive fragments, respectively. $\mathcal{F}$ is a correct fragmentation with respect to $\mathcal{C}$ iff both the following conditions hold:*

1. $I \cap (F_{NS} \cup F_S) = I$ (Completeness requirement),
2. $\forall c_i \in \mathcal{C}$: $(F_{NS} \cup F_S) \cup \mathcal{D} \not\models c_i$, otherwise $P(c_i) \leq 1/k$ (Privacy requirement).

According to Definition 7, a fragmentation is complete if taking the union of the fragments and removing the fake tuples, yields the original instance *I*. Also, it preserves privacy if applying the dependencies, as deduction rules, to the tuples in F_{NS} and F_S, as ground tuples, does not imply a privacy constraint. Otherwise, the probability of the knowledge inferable from the constraint should be equal to or less than 1/*k*.

3.2 *k*-Anonymity-Based Horizontal Fragmentation Algorithm

Based on Definition 7, we present an algorithm that produces a correct fragmentation. The general scheme of our method is showed in Figure 2. First, we identify those tuples in I that instantiate some privacy constraints (step 1). These tuples are called *explicitly sensitive tuples* and move them to F_S (step 2). We also identify the tuples in I that do not explicitly violate any constraint but implicitly violate some constraints, i.e., by applying dependencies. These tuples are called *implicitly sensitive tuples* and move them to F_S (step 2). By this step, all explicitly and implicitly sensitive tuples are moved to F_S but there may be some tuples that can be used to gain extra information about the sensitive tuples. Consider the tuples that their existence depends *only* on the existence of other tuples, i.e., produced by only applying dependencies to other tuples. These tuples are called *dangling tuples* and move them to F_S (step 2). After step 2, all remaining tuples in I are non-sensitive (because all sensitive tuples have been moved to F_S) and move them to F_{NS} (step 3). For the explicitly sensitive tuples in step 1, we produce some fake tuples (k - 1 violation sets) to k-anonymize their respective sensitive knowledge and place them in F_S (step 4). In addition to the fake violation sets, some other fake tuples is produced and placed in F_S (step 4), c.f. Section 3.3. Subsequently, we chase F_S to satisfy all dependencies (step 5, not shown in Figure 2).

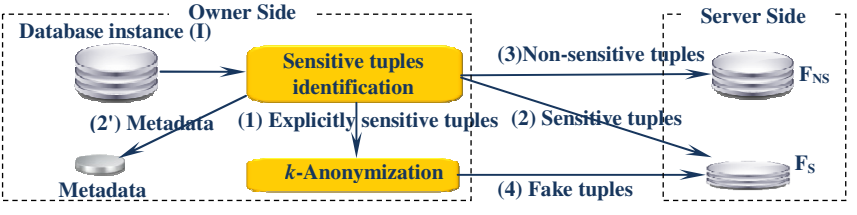

Fig. 2. General scheme of the proposed method

For each sensitive tuple in step 2 a metadata, e.g., a tuple id, is produced and placed in F_S along with that tuple (step 2′). These metadata are also stored at the owner side and used to recognize the real tuples in the future accesses to F_S.

3.3 Cascading Tuples

Consider tuples $t_1, t_2, \ldots, t_n$, where tuple t_n is generated by applying dependency d_{n-1} to tuple t_{n-1} and t_{n-1} itself is generated by applying dependency d_{n-2} to tuple t_{n-2}, … , finally tuple t_2 is generated by applying dependency d_1 to tuple t_1. We call this phenomenon *cascading tuples*. Let tuple t_n be explicitly sensitive. Therefore, tuples $t_1, t_2, \ldots, t_{n-1}$ are implicitly sensitive. According to the previous section, t_n is anonymized with some fake tuples and moved to F_S. Tuples $t_1, t_2, \ldots, t_{n-1}$ are also moved to F_S. The attacker observes that $t_1, t_2, \ldots, t_{n-1}$ are not explicitly sensitive but they are in F_S and if dependencies $d_1, d_2, \ldots, d_{n-1}$ be applied consecutively to these tuples, then t_n will be generated. Thus, s/he infers that $t_1, t_2, \ldots, t_{n-1}$ are implicitly sensitive tuples and consequently *real* tuples. Based on this observation, s/he infers

that t_n is also a *real* tuple with probability 1 that is greater than value $1/k$ in Definition 7, violating the privacy requirement. To thwart this inference, we propose the following solution:

For a set $\sum$ of TGDs we construct a directed graph, called *dependency graph*, denoted by DG, as follows:

- For each TGD $\forall \boldsymbol{x}\ (\phi(\boldsymbol{x}) \longrightarrow \exists \boldsymbol{y}\ \psi(\boldsymbol{x}, \boldsymbol{y}))$: for the set of atoms $a_1, a_1, \ldots, a_n$ in ϕ, add a new node to DG, if it does not already exist.
- For each TGD $\forall \boldsymbol{x}\ (\phi(\boldsymbol{x}) \longrightarrow \exists \boldsymbol{y}\ \psi(\boldsymbol{x}, \boldsymbol{y}))$: add an edge from the node corresponding to ϕ to the node that contain atom b of ψ.

Let t_n be anonymized with k - 1 fake tuples $f_1, f_2, \ldots, f_{k-1}$. For each f_i (i=1, …, k - 1) we find a node in DG from which f_i can be instantiated. Then, proceed downwards to a leaf node and instantiate with fake values (values from domains of attributes) all the nodes on the path to that leaf. In this way, the probability of the tuples $f_1, f_2, \ldots, f_{k-1}$, and t_n and their respective implicitly sensitive tuples being real, is equal to or less than $1/k$. Therefore, for each explicitly sensitive real tuple we produce some fake tuples in the above way to prevent from inferences about the real tuples.

3.4 *k*-Anonymity-Based Horizontal Fragmentation Algorithm

The proposed algorithm consists of the following steps:

Step 1. Initially, fragments F_S and F_{NS} are empty. A temporary set *fake_tuples* is considered that will contain the fake tuples used to anonymize the sensitive tuples.

Step 2. The purpose of this step is to identify the explicitly sensitive tuples and move them to F_S. For each constraint $\alpha(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ in $\mathcal{C}$, we identify vectors of constants $\boldsymbol{a}$ such that the instantiation $\alpha(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ of variables $\boldsymbol{x}$ with constants $\boldsymbol{a}$ holds in the input instance I, i.e., $I \vDash \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$. This implies that I explicitly violates the constraint (as a result, a sensitive knowledge can be inferred from I). To prevent this explicit inference, we remove from I and move to F_S the tuples participating in the instantiation of $\alpha(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$, i.e., the violation set of the constraint. Also, we anonymize these tuples with some fake tuples. The fake tuples are moved to *fake_tuples.*

Step 3. The purpose of this step is to identify the implicitly sensitive and dangling tuples and move them to F_S. By removing the explicitly sensitive tuples from I, some dependencies may be violated by I, helping the attacker to gain extra information about the sensitive tuples. To prevent this, for each TGD $\forall \boldsymbol{x}\ (\phi(\boldsymbol{x}) \longrightarrow \exists \boldsymbol{y}\ \psi(\boldsymbol{x}, \boldsymbol{y}))$ we identify those tuples of constants $\boldsymbol{a}, \boldsymbol{b}$ such that the instantiations $\alpha(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ and $\psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}]$ of variables $\boldsymbol{x}, \boldsymbol{y}$ with constants $\boldsymbol{a}, \boldsymbol{b}$ hold in the union of the input instance and sensitive fragment. If both of the body and head are only in I or only in F_S, no inference can be done. Otherwise, the following inferences may be possible:

Inference type 1: The TGD body is in I and the TGD head is in F_S:

$$I \vDash \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}],\ I \nvDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}],\ F_S \vDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}]$$

In this case, an attacker can apply the TGD, produce its head, and implicitly infer about some sensitive tuples, i.e., the head of TGD, in F_S. To prevent this implicit inference, all parts of the TGD body must be moved to F_S.

Inference type 2: The TGD body is in F_S and the head is in I:

$$F_S \vDash \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}] \text{ , } F_S \nvDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}], I \vDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}]$$

In this case, an attacker observes that there exist some tuples that are generated by the TGD in I but the tuples in the TGD body are missing in I (dangling tuples), because they are moved to F_S. To prevent this inference, all parts of the TGD head must be moved to F_S.

Inference type 3: Some parts of the body are in I and some parts of the head are in F_S:

$$I \cup F_S \vDash \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}], I \cup F_S \vDash \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}, \boldsymbol{b}/\boldsymbol{y}]$$

This case is a general type of the two previous types. In this case, all parts of the body and all parts of the head that are in I must be moved to F_S.
If any tuple is moved to F_S in this step, we will repeat this step until there exists no tuple that can be used for one or more of the above three types of inferences.

Step 4. By this step, there is no tuple in I that can be used for inference. Now, we move the remaining tuples, i.e., non-sensitive tuples, to F_{NS}.

Step 5. We add the fake tuples produced in Step 1, i.e., tuples in *fake_tuples*, to F_S. The newly added tuples may cause F_S violating some dependencies. In this case, we chase F_S with the dependencies and produce a chased fragment satisfying all dependencies. In the newly generated tuples, we replace the "labeled nulls" (see [9]) with some fake values (values from domains of attributes). This prevents the attacker from knowing that they are produced from some fake tuples.

Step 6. Finally, fragmentation $\mathcal{F}$ consisting of F_S and F_{NS} is returned as the result of the fragmentation algorithm.

Illness_F_{NS}

Name	Disease
Andy	Hypertension
Alice	Obesity
David	Heart disease
Linda	Cholera

Treatment_F_{NS}

Name	Medicine
Andy	Med A
Alice	Med B
David	Med C

Info_F_{NS}

Name	DoB	ZIP
Andy	1981/01/03	94142
Alice	1953/10/07	86342
David	1999/01/20	20688
Linda	1989/01/03	94139

Illness_F_S

Name	Disease
Bob	Aids
Jim	Aids
Sara	Influenza
Sara	Flu
Tom	Viral disease
Mary	**Viral disease**

Treatment_F_S

Name	Medicine
Bob	Med D
Bob	Med E
Tom	Med X
Tom	Med Y
Mary	Med X
Mary	Med Y

Info_F_S

Name	DoB	ZIP
Bob	1952/02/12	79232
Sara	2000/01/20	40496
Tom	1970/11/01	23567
Jim	*1959/02/15*	*78452*
Mary	**1972/04/03**	**63234**

Fig. 3. A fragmentation for instance I in Figure 1

EXAMPLE 6. Consider the instance I, the set of dependencies $\mathcal{D}$, the set of privacy constraints $\mathcal{C}$, and the set of anonymization rules $\mathcal{A}$, from Examples 1-5, respectively. We fragmented I using our algorithm. The result is shown in Figure 3. Attribute values in bold represent the tuples produced for cascading tuples. Attribute values in italic represent the labeled nulls replaced with fake values; the chase applied the dependency d_3 to the fake tuple *Illness*(Jim, Aids) and generated *Info*(Jim, labeled_null$_1$, labeled_null$_2$) and replaced them with values 1959/02/15 and 78452, respectively. □

4 Conclusion

We presented a horizontal fragmentation that outsources all data but do not use encryption to provide privacy. Also, an algorithm presented that produces a correct fragmentation. The main application of the method is for scenarios where even encrypting or suppressing the sensitive data cannot guarantee the privacy.

Several issues remain: our method, to achieve k-anonymity, produces k - 1 fake violation sets for each violated constraint. This may result in high storage and bandwidth overheads for large volumes of sensitive data and large values of k. The adoption of other ways, such as a probabilistic approach, to provide privacy with degree k but with less than k - 1 fake violation sets, can be investigated as a future work. In this paper, we employed a version of the chase, called *standard chase*, which put some restrictions on the dependencies and constrains, such as being positive and conjunctive. Investigating the applicability of other versions of the chase in the method can be studied further. The anonymity principle of our method has some similarities to the l-diversity privacy model [11]. Investigating other privacy models, such as t-closeness [12], to provide a stronger privacy model for the proposed method can be valuable.

References

1. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Enforcing Confidentiality Constraints on Sensitive Databases with Lightweight Trusted Clients. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security 2009. LNCS, vol. 5645, pp. 225–239. Springer, Heidelberg (2009)
2. Wiese, L.: Horizontal Fragmentation for Data Outsourcing with Formula-Based Confidentiality Constraints. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) IWSEC 2010. LNCS, vol. 6434, pp. 101–116. Springer, Heidelberg (2010)
3. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a Few: Outsourcing Data While Maintaining Confidentiality. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 440–455. Springer, Heidelberg (2009)
4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security 13, 1–33 (2010)

5. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: A distributed architecture for secure database services. In: Second Biennial Conference on Innovative Data Systems Research, pp. 186–199 (2005)
6. Foresti, S.: Preserving privacy in data outsourcing. Springer-Verlag New York Inc. (2011)
7. Beeri, C., Vardi, M.Y.: A Proof Procedure for Data Dependencies. J. ACM 31, 718–741 (1984)
8. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. 4, 455–469 (1979)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science 336, 89–124 (2005)
10. Samarati, P., Sweeney, L.: Generalizing data to provide anonymity when disclosing information (abstract). In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, p. 188. ACM, Seattle (1998)
11. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: l-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data 1, 3 (2007)
12. Ninghui, L., Tiancheng, L., Venkatasubramanian, S.: t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In: 23rd International Conference on Data Engineering, pp. 106–115 (2007)