Horizontal Fragmentation for Data Outsourcing with Formula-Based Confidentiality Constraints

Lena Wiese

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
lena.wiese@udo.edu

Abstract. This article introduces the notion of horizontal fragmentation to the data outsourcing area. In a horizontal fragmentation, rows of tables are separated (instead of columns for vertical fragmentation). We give a formula-based definition of confidentiality constraints and an implication-based definition of horizontal fragmentation correctness. Then we apply the chase procedure to decide this correctness property and present an algorithm that computes a correct horizontal fragmentation.

1 Introduction

The interest in outsourcing data to a third-party storage ("server") site has increased over the last decade with the main advantage being the reduction of storage requirements at the local ("owner") site. Yet, because the storage server usually cannot be fully trusted, several approaches to protect the outsourced data have emerged. In general, there are the following approaches:

- **Encryption only:** Before outsourcing, all data tuples are encrypted [1, 2]; query execution on the outsourced data is difficult and inexact.
- **Vertical fragmentation and encryption:** Some table columns are separated into different fragments as cleartext while other (partial) tuples are encrypted [3, 4]; query execution is easier on the cleartext part but still decryption has to be executed by the data owner to execute queries on the encrypted part.
- **Vertical fragmentation only:** When the data owner is willing to store some columns at his trusted local site in an owner fragment, other columns can be outsourced safely in a server fragment [5, 6]; the fragmentation can be optimized with respect to assumptions on query frequencies.

In this article we refrain from using encryption. It has already been argued in [5] that encryption is not necessary if a fragmentation is identified of which one fragment is stored at the trusted owner site. We reinforce the point that encryption is costly as it requires key management and long-term security of the encryption scheme. Moreover, often querying encrypted data is suboptimal [2] or only weak encryption is possible [7].

In this article we adopt the client-server setting of [5]. In their approach for vertical fragmentation, only projection onto columns is supported and thus the so called "confidentiality constraints" are merely defined as sets of attributes of the database schema. They do not take into account the content – the actual data values – in a database instance. Moreover this approach only considers one ("universal") relation instead of a full-blown database schema with several relation symbols (which is usually the case for databases in some normal form). However, normalized databases are advantageous because they reduce storage requirements (by removing redundancy) and facilitate data management (e.g., by avoiding anomalies). In the same sense, vertical fragmentation also lacks the notion of database dependencies – that is, constraints that can be specified on the database relations. Such database dependencies are crucial when it comes to controlling *inferences*: with dependencies further information can be derived from some (partial) database entries.

To extend the "vertical fragmentation only" approach we make the following contributions:

- We propose to use not only vertical but also horizontal fragmentation. In particular, we aim to filter out confidential *rows* to be securely stored at the owner site. The remaining rows can safely be outsourced to the server.
- We extend expressiveness of the "confidentiality constraints" by using first-order formulas instead of sets of attribute names. This implies that vertical fragmentation can be data-dependent in the sense that only some cells of a column have to be protected.
- We explicitly allow a full database schema with several relations symbols and a set of database dependencies. With these dependencies we introduce the possibility of inferences to the fragmentation topic and provide an algorithm to avoid such inferences.

The paper is organized as follows. Section 2 sets the basic terminology. Section 3 introduces logical formulas as syntactical material for confidentiality constraints. Section 4 presents a definition for horizontal fragmentation correctness; it analyzes the problem of fragmentation checking and introduces a new algorithm for the computation of a correct fragmentation. Lastly, we argue that also vertical and hybrid fragmentation can be data-dependent (in Section 5) and conclude this article in Section 6.

2 System Description

We view relational databases using the formalism of first-order predicate logic with equality (and possibly with other built-ins like comparison operators). A database has a **schema** $DS = \langle \mathcal{P}, \mathcal{D} \rangle$ where $\mathcal{P}$ is the set of relation symbols (that is, table names) and $\mathcal{D}$ is the set of dependencies between the relations. Each relation symbol P comprises a set of attributes (that is, data columns) and the arity $\mathit{arity}(P)$ is the number of attributes of relation symbol P. Each such attribute has associated a domain of constant values. A database **instance**

is a set of ground atomic formulas (representing data tuples or data rows) in accordance with the database schema; these are formulas without variables and each formula consists of one relation symbol that is filled with some appropriate constant values.

As database dependencies $\mathcal{D}$ we allow tuple-generating dependencies (tgds) and equality generating dependencies (egds). Tuple-generating dependencies can contain both universal and existential quantifiers. Their body as well as their head consists of a conjunction of atomic formulas.

Definition 1 (Dependencies). *A* tuple-generating dependency (tgd) *is a closed formula of the form*

$$\forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$$

where $\phi(\boldsymbol{x})$ and $\psi(\boldsymbol{x}, \boldsymbol{y})$ are conjunctions of atomic formulas. $\phi(\boldsymbol{x})$ is called the body *and $\exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$ is called the* head *of the tgd.*
A tgd is called full *if there are no existentially quantified variables $\boldsymbol{y}$ in ψ.*
An equality-generating dependency (egd) *is a closed formula of the form*

$$\forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow (x = x')$$

where $\phi(\boldsymbol{x})$ is a conjunction of atomic formulas and x and x' are contained in $\boldsymbol{x}$.

Note that a tgd indeed consists of disjunctions and negations (as the material implication $\rightarrow$ is only an abbreviation for disjunction and negation); and tgds can easily be written in conjunctive normal form when distributing the conjuncts of the head over the disjunctions in the body.

For formulas that are more general than tgds (for example, arbitrary disjunctions) feasibility of fragmentation problems cannot be ensured. More precisely, decidability for the "fragmentation checking" and "fragmentation computation" problems (see Section 4) cannot be established in general and the chase procedure as well as the search algorithm presented below are not guaranteed to terminate. For tgds, cyclicity also leads to undecidability; this is why we provide the additional restriction of weak acyclity in Definition 2. But before doing that we introduce a running example.

Example 1. In our running example, the database comprises some medical records and the relation symbols are $\mathcal{P} = \{\mathit{Illness}, \mathit{Treatment}\}$. The relation *Illness* has the two attributes (that is, column names) *Name* and *Disease*; the relation *Treatment* has the two attributes *Name* and *Medicine*. An instance of the database schema with these relation symbols would be

I:

Illness	*Name*	*Diagnosis*
	Mary	Aids
	Pete	Flu
	Lisa	Flu

Treatment	*Name*	*Medicine*
	Mary	MedA
	Mary	MedB
	Pete	MedA

The set of dependencies $\mathcal{D}$ are tgds or egds. It can for example contain a formula that states whenever a patients takes two specific medicines, then he is certainly ill with the disease aids: $\forall x\, \mathit{Treatment}(x, \mathsf{MedA}) \wedge \mathit{Treatment}(x, \mathsf{MedB}) \rightarrow$

Illness(x, Aids). This is a full tgd. Or it can contain a tgd that states that if a patient receives medical treatment there should be an appropriate diagnosis: $\forall x, y\ \mathit{Treatment}(x, y) \rightarrow \exists z \mathit{Illness}(x, z)$. An egd could be a key dependency or a functional dependency if for example a patient ID uniquely determines a patient's name.

On tgds we now pose the additional requirement of weak acyclicity (see [8]). This property avoids that there are cyclic dependencies among existentially quantified variables. Such cyclicity could possibly lead to the case that database instances satisfying these dependencies are infinite which would make the system infeasible. Weakly acyclic tgds have nice properties as will be used later on; for example, the chase on weakly acyclic tgds is ensured to run in polynomial time.

Definition 2 (Weak acyclicity [8]). *For a given set S of tgds, its* dependency graph *is determined as follows:*

- *For each relation symbol P occurring in S, create $\mathit{arity}(P)$ many nodes $P_1, \ldots, P_{\mathit{arity}(P)}$; these are the* positions *of P.*
- *For every tgd $\forall \boldsymbol{x}\,(\phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y}))$ in S: if a universally quantified variable $x \in \boldsymbol{x}$ occurs in a position P_i in ϕ and in a position P'_j in ψ, add an edge from P_i to P'_j (if it does not already exist).*
- *For every tgd $\forall \boldsymbol{x}\,(\phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y}))$ in S: if a universally quantified variable $x \in \boldsymbol{x}$ occurs in a position P_i in ϕ and in a position P'_{j_1} in ψ, and an existentially quantified variable $y \in \boldsymbol{y}$ occurs in a position P''_{j_2} in ψ, add a* special edge *marked with $\exists$ from P_i to P''_{j_2} (if it does not already exist).*

A dependency graph is weakly acyclic*, iff it does not contain a cycle going through a special edge. We call a set of tgds weakly acyclic whenever its dependency graph is weakly acyclic.*

In our example, the two tgds are acyclic (and hence also weakly acyclic) because edges only go from *Treatment* to *Illness*.

For an open formula $\phi(\boldsymbol{x})$ (with free variables $\boldsymbol{x}$) we can identify **instantiations** in an instance I; this corresponds to an evaluation of ϕ in I if ϕ is seen as a query: we find those constant values $\boldsymbol{a}$ (in accordance with the domains of the attributes) that can be substituted in for the variables $\boldsymbol{x}$ (written as $[\boldsymbol{a}/\boldsymbol{x}]$) such that $\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ holds in the instance I. For example, evaluating the formula $\mathit{Treatment}(x, \mathsf{MedA}) \wedge \mathit{Treatment}(x, \mathsf{MedB})$ would substitute in Mary for $\boldsymbol{x}$ and result in the answer $\mathit{Treatment}(\mathsf{Mary}, \mathsf{MedA}) \wedge \mathit{Treatment}(\mathsf{Mary}, \mathsf{MedB})$.

Our aim is now to decompose an input instance I into two disjoint sets of tuples: the "server fragment" F_s and the "owner fragment" F_o. The server fragment has to be such that it satisfies the notion of "fragmentation correctness" (see Definition 4 below) even though we assume that the server has full (a priori) **knowledge** of the database dependencies $\mathcal{D}$. This can be seen as a form of the "honest but curious" attacker model that is often used in cryptographic settings.

3 An Extended Syntax for Confidentiality Constraints

Usually for vertical fragmentation (see [3–5]) a confidentiality constraint is just a subset of the attributes of a universal relation. Its meaning is that no combination of values (a subtuple of the universal relation with all attributes of the constraint) must be fully disclosed. For example, for the relation *Illness* the confidentiality constraint $\{Name, Disease\}$ means that no full tuple of *Illness* must be disclosed; but either the *Name* column or the *Disease* column may appear in a secure fragment. The singleton constraint $\{Name\}$ means that the *Name* column must be protected but the *Disease* column can be published. In other words, a confidentiality constraint is satisfied, if of all the attributes in the constraint either one attribute is encrypted in the outsourced relation or the universal relation is decomposed such that each outsourced fragment is missing at least one of the attributes involved in the confidentiality constraint.

We now introduce the formula-based notation for confidentiality constraints that will be used throughout this article. Attribute-based confidentiality constraints for vertical fragmentation can be expressed by formulas with free variables: the free variables of a formula are those contained in the confidentiality constraint. For example, the confidentiality constraint $\{Name, Disease\}$ as a formula will be written as $Illness(x, y)$. The variable x for the attribute *Name* as well as the attribute y for the attribute *Disease* are free such that either column can be removed (to yield a secure fragment) or encrypted. Other attributes not involved in a confidentiality constraint are written as existentially quantified (hence bound) variables. For example, the singleton constraint $\{Name\}$ will be expressed as $\exists y\, Illness(x, y)$: the only free variable is x and hence the *Name* column must be protected.

Going beyond the attribute-based confidentiality constraints used in prior work, we now state how formula-based constraints can greatly improve expressiveness of constraints; hence, formulas make it possible to express finer-grained confidentiality requirements:

1. We can easily express protection of whole relations by existentially quantifying all variables instead of using several singleton constraints. For example $\exists xy\, Illness(x, y)$ expresses that the whole relation *Illness* must not be outsourced to the server.
2. We can express data-dependent constraints by using constant values. For example, $\exists x\, Illness(x, \mathsf{Aids})$ signifies that no row with the value Aids for the attribute *Disease* must be outsourced. In contrast, the open formula $Illness(x, \mathsf{Aids})$ signifies that no combination of a patient name with the disease aids must be outsourced; that is, all patient names of those rows with an aids entry must be protected. This makes a difference when hybrid fragmentation is used where vertical and horizontal fragmentation can be combined.
3. We can combine several atomic expressions (expressions with one relation symbol only) in formulas with logical connectives like conjunction. For example, $Illness(x, \mathsf{Aids}) \wedge Treatment(x, \mathsf{MedB})$ means that for patients suffering

from aids and at the same time being treated with a particular medicine MedB, either the name column from the relation *Illness* or from the relation *Treatment* must be suppressed. For the formula $\exists y(\mathit{Illness}(x, \mathsf{Aids}) \wedge \mathit{Treatment}(x, y))$ the same applies for any medicine whatsoever.
With our semantics, protection of a disjunctive formula (like for example $\exists x(\mathit{Illness}(x, \mathsf{Aids}) \vee \mathit{Illness}(x, \mathsf{Cancer}))$) can be simulated by splitting the formula into separate constraints ($\exists x \mathit{Illness}(x, \mathsf{Aids})$ and $\exists x \mathit{Illness}(x, \mathsf{Cancer})$): a server not allowed to know the whole disjunction is also not allowed to know any of the single disjuncts. In other words, each single disjunct implies the whole disjunctions.

We define formula-based confidentiality constraints – that can be used for horizontal as well as hybrid fragmentation – as formulas that use the syntactic material (relation symbols and constants) of the database schema; we restrict the syntax to formulas without negation ("positive formulas") that use only conjunction $\wedge$ as a logical connective and have possibly some variables bound by existential quantifiers in a prefix.

Definition 3 (Formula-based confidentiality constraints). *Formula-based confidentiality constraints are positive conjunctive formulas possibly with existential prefix that mention only relation symbols and constants from the domains of the attributes as defined by the database schema.*

Free variables will only be used for vertical fragmentation. In the next section we concentrate on horizontal fragmentation. In this case we restrict confidentiality constraints to "closed" formulas; that is, all variables will be existentially quantified. In sum, a set of formula-based confidentiality constraints for horizontal fragmentation corresponds to a union of conjunctive Boolean queries. Another result of [8] that we will use is that certain answers for unions of conjunctive queries can be computed in polynomial time.

4 Horizontal Fragmentation

For *vertical* fragmentation, fragments consist of some cleartext columns and some encrypted (partial) tuples. In [5], the server and the owner fragment can simply be represented by two disjoint sets of attribute names. The natural join $\bowtie$ is used for reconstruction of the original relation (or parts of the original relation after a query; see [3]). In [5], the join of the server and the owner fragment has to be computed only on the tuple id because of an additional non-redundancy requirement. Previous work for vertical fragmentation covers the following two requirements called "fragmentation correctness": **completeness** (that is, the original relation can be reconstructed by the owner from the fragments) and **confidentiality** (not all attributes of an attribute-based confidentiality constraint are contained in one fragment). In the "fragmentation only" approach [5] the requirement of **non-redundancy** (each attribute is contained either in the server or the owner fragment) is added; this concept has not been analyzed

for approaches involving encryption because encrypted tuples usually contain redundant information.

In contrast, in our *horizontal* fragmentation approach fragments are sets of rows instead of sets of columns. The fragments (the rows in the server and the owner fragment) have to be combined again by simply taking the union $\cup$ of the fragments. We now introduce our notion of fragmentation correctness for horizontal fragmentation. The **completeness** requirement easily translates to horizontal fragmentation by requiring that the union of the fragments (that is, rows) yields the original database instance. In the same sense, **non-redundancy** means that no row is contained in both the server and the owner fragment. The **confidentiality** requirement is more complex than in the vertical case because

- it depends on the data in the database instance and not only on the attribute names
- it involves the database dependencies that are assumed to be known a priori by the server
- it respects the logical nature of closed formula-based confidentiality constraints.

Hence we base confidentiality on the notion of logical implication $\models$. A set S of formulas implies a formula f (written $S \models f$) if and only if every model (that is, every satisfying interpretation) of S also satisfies f. If the server knows some dependencies between data – as for example the database dependencies $\mathcal{D}$ – these can be applied as deduction rules on the server fragment to infer other facts that are presumably protected in the client fragment. In our system we have a strong attacker model in the sense that we assume the server to be aware of all dependencies in $\mathcal{D}$ and hence the server fragment has to be such that application of these dependencies do not enable the inference of any of the confidentiality constraints. We can thus say that a fragmentation ensures confidentiality if the server fragment (treating each tuple as a ground atomic formula) and the database dependencies (that can be applied as deduction rules) do not imply any formula-based confidentiality constraint.

We adapt Definition 2 of [5] to formula-based confidentiality constraints as follows. Note that our fragments are sets of tuples (that is ground atomic formulas) in contrast to [5] where the fragments are sets of attribute names. Also note that for horizontal fragmentation we only accept closed confidentiality constraints as already mentioned in Section 3.

Definition 4 (Horizontal fragmentation correctness). *Let I be an instance of a database schema $DS = \langle \mathcal{P}, \mathcal{D} \rangle$, $\mathcal{C} = \{c_1, \ldots, c_m\}$ be set of closed formula-based confidentiality constraints, and $\mathcal{F} = \{F_o, F_s\}$ be a fragmentation of I, where F_o is stored at the owner and F_s is stored at a storage server. $\mathcal{F}$ is a* correct horizontal fragmentation *with respect to $\mathcal{C}$, iff: 1) $F_o \cup F_s = I$ (completeness); 2) for every $c_i \in \mathcal{C}$, $F_s \cup \mathcal{D} \not\models c_i$ (confidentiality); 3) $F_o \cap F_s = \emptyset$ (non-redundancy).*

Our aim is now twofold: we first analyze how a given fragmentation can be checked for correctness and then elaborate how a correct fragmentation can be computed from an input instance.

4.1 Fragmentation Checking

We now analyze the following problem:

Problem 1. Given a database schema $DS = \langle \mathcal{P}, \mathcal{D} \rangle$, an instance I of DS, a set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of closed formula-based confidentiality constraints, and a fragmentation $\mathcal{F} = \{F_o, F_s\}$, the *fragmentation checking* problem is to decide whether $\mathcal{F}$ is a correct horizontal fragmentation of I.

Correctness and non-redundancy requirements of Definition 4 can be checked easily by the owner. However checking confidentiality again is more complex. We have to check that F_s does not reveal any confidentiality constraint itself; neither should F_s imply any confidentiality constraint whenever the server applies the database dependencies to the server fragment. So in general, it might happen that the server fragment F_s does not satisfy the database dependencies and the server uses them to deduce other facts. To ensure that the deduced facts do not breach confidentiality of the confidentiality constraints, the owner has to apply the dependencies to the server fragment to check the confidentiality requirement. We will use results of the "data exchange" area to decide the fragmentation checking problem.

The famous *chase* procedure was introduced as a method to decide implication between two sets of dependencies [9]. Later on, it was used in [8] to compute "universal solutions" in a data exchange setting and in [10] for database repair checking. From a confidentiality point of view it was used in [11] to extend a mandatory access control (MAC) system and mentioned in [12] as a method to decide security of view instances. In particular, the results of [8] show that for the wide class of weakly acyclic tuple-generating dependencies and equality-generating dependencies (see Definition 2), the chase computes a universal solution containing some "null values" in time polynomial in the size of the input instance. It is also shown in [8] that if a conjunctive query is evaluated in a universal solution, this evaluation can also be done in polynomial time and the result is the set of "certain answers": those answers that hold in every possible data exchange solution of the input instance.

The results of [8] can be used to check confidentiality of constraints in a server fragment F_s as follows. If we restrict the database dependencies $\mathcal{D}$ to be weakly acyclic tgds and egds, the chase on the server fragment F_s terminates in time polynomial in the size of the server fragment. It results in a chased server fragment containing null values: existentially quantified variables in tgds are filled in with new null values and egds are applied to equate some values. More formally, if there is a mapping (a homomorphism) from the variables in the body of a dependency to the constants $const(F_s)$ and the null values $nulls(F_s)$ in the server fragment, then a chase step can be executed ("applied"). See also [8, 9, 11] for details.

Definition 5 (Application of dependencies). *A tgd* $\forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$ *can be* applied *to the server fragment* F_s *if*

- *there is a homomorphism* $h : \boldsymbol{x} \rightarrow const(F_s) \cup nulls(F_s)$ *such that for every atom* $P(x_1, \ldots, x_k)$ *(where the free variables are* $x_i \in \boldsymbol{x}$ *for* $i = 1 \ldots k$*) in* $\phi(\boldsymbol{x})$*, the atom* $P(h(x_1), \ldots, h(x_k))$ *is contained in* F_s
- *but* h *cannot be extended to map the existentially quantified variables* $\boldsymbol{y}$ *in the head* $\exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$ *to* $const(F_s) \cup nulls(F_s)$ *such that for every atom* $Q(x_1, \ldots, x_l, y_1, \ldots, y_{l'})$ *(where the free variables are* $x_i \in \boldsymbol{x}$ *for* $i = 1 \ldots l$ *and* $y_j \in \boldsymbol{y}$ *for* $j = 1 \ldots l'$*) in* $\psi(\boldsymbol{x}, \boldsymbol{y})$*, the atom*

$$Q(h(x_1), \ldots, h(x_l), h(y_1), \ldots, h(y_{l'}))$$

  is contained in F_s*.*

The result of applying a tgd to F_s *is the union of* F_s *and all those atoms that can be generated from all atoms* $Q(x_1, \ldots, x_l, y_1, \ldots, y_{l'})$ *of* $\psi(\boldsymbol{x}, \boldsymbol{y})$ *with the variables* $\boldsymbol{x}$ *mapped according to* h *and the variables* $\boldsymbol{y}$ *each mapped to a new null value. An egd* $\forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow (x = x')$ *can be* applied *to the server fragment* F_s *if*

- *there is a homomorphism* $h : \boldsymbol{x} \rightarrow const(F_s) \cup nulls(F_s)$ *such that for every atom* $P(x_1, \ldots, x_k)$ *in* $\phi(\boldsymbol{x})$*, the atom* $P(h(x_1), \ldots, h(x_k))$ *is contained in* F_s
- *but* $h(x) \neq h(x')$*.*

The result of applying an egd to F_s *is obtained by*

- *replacing all occurences of the null value in* F_s *with the constant if one of* $h(x)$ *and* $h(x')$ *is a null value and the other is a constant or*
- *replacing all occurences of one null value in* F_s *with the other if both* $h(x)$ *and* $h(x')$ *are null values.*

Note that because we assume that the server fragment F_s is a subset of the input instance I and I is assumed to satisfy the dependencies, chasing with an egd cannot "fail" (that is, lead to an inconsistency).

On the chased fragment the notion of "certain answers" can also be defined: a certain answer to a query is one that holds in any possible fragment that contains F_s as a subset and that satisfies the database constraints $\mathcal{D}$; and we can find the certain answers by posing a query to the chased server fragment. Because we defined confidentiality constraints to be positive, existential, conjunctive and closed formulas, when we pose a constraint as a query to the chased server fragment, the certain answers can be computed in polynomial time as shown in [8]. We can be sure that confidentiality of a constraint is preserved if the certain answer of this constraint in the chased server fragment is *false*. We give a small example to illustrate the procedure.

Example 2. Assume that we have given the server fragment

F_s:

Illness	*Name*	*Diagnosis*
	Lisa	Flu

Treatment	*Name*	*Medicine*
	Mary	MedB
	Mary	MedA
	Pete	MedC

The set of dependencies contains two tgds that link treatments with diseases:

$$\mathcal{D} = \{\forall x\ \mathit{Treatment}(x, \mathsf{MedC}) \rightarrow \exists z \mathit{Illness}(x, z),$$
$$\forall x\ \mathit{Treatment}(x, \mathsf{MedA}) \wedge \mathit{Treatment}(x, \mathsf{MedB}) \rightarrow \mathit{Illness}(x, \mathsf{Aids})\}$$

Chasing F_s with $\mathcal{D}$ results in the following instance where τ is a null value:

F_{chase}:

Illness	*Name*	*Diagnosis*
	Lisa	Flu
	Pete	τ
	Mary	Aids

Treatment	*Name*	*Medicine*
	Mary	MedB
	Mary	MedA
	Pete	MedC

Assume the confidentiality constraints stating that it should not be outsourced that there is a patient with aids and that there is a disease from which patient Pete suffers:

$$\mathcal{C} = \{\exists x\ \mathit{Illness}(x, \mathsf{Aids}),$$
$$\exists y\ \mathit{Illness}(\mathsf{Pete}, y)\}$$

We see that the certain answers of the two confidentiality constraints in F_{chase} are both *true* and hence the server fragment does not comply with the confidentiality requirements. In this case the server fragment should not be outsourced.

In addition to fragmentation correctness, the server fragment should be maximal and the owner fragment minimal in some sense; for example, the storage requirements at the owner site should be minimized. Beyond storage analysis, the metrics in [5] also analyze query frequencies. In the context of database repairs, [10] survey and analyze other optimization criteria that can also be adopted for fragmentation approaches.

4.2 Fragmentation Computation

We now propose an algorithm for a set of database dependencies $\mathcal{D}$ containing weakly acyclic tgds and egds and a set of closed confidentiality constraints $\mathcal{C}$. The main idea is the following: starting with the original input instance I we identify tuples that must be moved from I to the owner fragment F_o or to the server fragment F_s by evaluating the confidentiality constraints and database dependencies as queries in I. The algorithm will decide for each affected tuple, whether it is possible to move it to the server fragment or not. The remaining tuples (not affected by the constraints and dependencies) can simply be moved to the server fragment.

The decision can be accompanied by several optimization criteria (like the ones mentioned previously in Section 4.1). In contrast to these, we propose here

to minimize the number $cells(F_o)$ of table cells that are moved to the owner fragment. That is, we take into account the size of the tuples where size is measured as the number of attributes. This indeed has an impact when several relations are contained in the database schema (in contrast to the approaches considering only a universal relation).

SEARCH:
- Input: instance I, confidentiality constraints $\mathcal{C}$, dependencies $\mathcal{D}$
- Output: correct horizontal fragmentation $\mathcal{F} = \{F_o, F_s\}$

1. $Inst = \emptyset$
2. for each $\exists \boldsymbol{x}\, \phi(\boldsymbol{x}) \in \mathcal{C}$: remove $\exists \boldsymbol{x}$
3. $Inst = Inst \cup \{\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}] \mid I \cup F_s \models \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]\}$
4. for each $\forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
5. $Inst = Inst \cup \{\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}] \mid I \cup F_s \models \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ AND $I \cup F_s \not\models \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}]\}$
6. if $Inst = \emptyset$: $F_s = F_s \cup I$; return $\mathcal{F} = \{F_o, F_s\}$
7. else choose $l_1 \wedge \ldots \wedge l_k \in Inst$
8. if $\{l_1, \ldots, l_k\} \subseteq F_s$: conflict
9. else choose $l_i \in \{l_1, \ldots, l_k\}$ such that $l_i \in I$
10. $F_o = F_o \cup l_i$; $I = I \setminus l_i$; SEARCH
11. $F_s = F_s \cup l_i$; $I = I \setminus l_i$; SEARCH

Fig. 1. Horizontal fragmentation algorithm

We now describe our algorithm in detail and provide a pseudocode listing in Figure 1. We start with the input instance I and $F_o = F_s = \emptyset$. We then take the confidentiality constraints $\mathcal{C} = \{c_1, \ldots, c_m\}$ and execute the following steps.

1. Remove all existential prefixes from constraints $c_i = \exists \boldsymbol{x}\, \phi(\boldsymbol{x})$ such that they are now open formulas $\phi(\boldsymbol{x})$ with free variables $\boldsymbol{x}$.
2. Evaluate the constraints in $I \cup F_s$. That is, find those tuples of constants $\boldsymbol{a}$ such that the instantiation $\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ of variables $\boldsymbol{x}$ with constants $\boldsymbol{a}$ holds in the input instance and the server fragment: $I \cup F_s \models \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$.
3. Add each such instantiation to the set of "candidate instantiations" $Inst$.

Similarly, we treat the database dependencies $\mathcal{D} = \{d_1, \ldots, d_m\}$ – with the difference that we have to find those instantiations for which the body of the dependency is satisfied but the head is not. Note that this will only apply to tgds: all egds are satisfied in I; they will never be violated in F_s which is a subset of I. Hence let d_i be a tgd: $d_i = \forall \boldsymbol{x}\, \phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$ where $\phi(\boldsymbol{x})$ is the body, $\exists \boldsymbol{y} \psi(\boldsymbol{x}, \boldsymbol{y})$ is the head and both are conjunctions of atomic formulas.

1. Evaluate each tgd in $I \cup F_s$ and find those instantiations such that the body is satisfied but the head is not. That is, find those tuples of constants $\boldsymbol{a}$ such that
   (a) the instantiation of the body $\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ of variables $\boldsymbol{x}$ with constants $\boldsymbol{a}$ holds in the input instance and the server fragment: $I \cup F_s \models \phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$.

(b) but the instantiated head $\exists \boldsymbol{y}\psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}]$ is *false* in $I \cup F_s$; that is, $I \cup F_s \not\models \exists \boldsymbol{y}\psi(\boldsymbol{x}, \boldsymbol{y})[\boldsymbol{a}/\boldsymbol{x}]$. Note that this is a closed formula.

2. Add the instantiated body $\phi(\boldsymbol{x})[\boldsymbol{a}/\boldsymbol{x}]$ to the set of candidate instantiations *Inst*.

The candidate set *Inst* contains only positive conjunctive ground formulas of the form $l_1 \wedge \ldots \wedge l_k$. In order to achieve consistency of the server fragment F_s with the database dependencies $\mathcal{D}$ without violating the confidentiality constraints $\mathcal{C}$, at least one of the conjuncts l_i has to be moved to the owner fragment F_o. Hence, if there is a formula in *Inst* for which all ground atoms $l_1, \ldots, l_k$ are contained in the server fragment, a conflict with the dependencies and constraints has occurred. The search then continues with a distinct subproblem by backtracking. Otherwise, choose one formula from *Inst* and one ground atom l_i of that formula that is contained in I (and hence neither contained in F_o nor F_s). Create two new subproblems: one by moving the ground atom l_i to F_o and the other one by moving l_i to F_s and recursively executing the search procedure on it. The candidate set *Inst* is emptied in every recursion. Repeat these steps until the evaluations of constraints and dependencies do not result in further candidate formulas; that is, until the candidate set *Inst* remains empty. Move all atoms remaining in I to the server fragment. The search indeed is a depth-first search along a binary tree as pictured in Figure 2.

Two additional operations can speed up the search process significantly: unit propagation and branch-and-bound pruning. Unit propagation means that a candidate formula consisting of a single ground atom can be moved to the owner fragment without trying to move it to the server fragment; moving it to the server fragment would immediately result in a conflict. The same applies to formulas in the candidate set *Inst* for which exactly one ground atom l_i is contained in I and all other ground atoms were already moved to the server fragment. Branch-and-bound pruning is helpful when an optimization requirement has to be fulfilled. In this case, not the first solution is output; instead, the search continues and tries to find a better one. We propose to count the number of table cells $\mathit{cells}(F_o)$ that are contained in the owner fragment F_o and try to minimize this amount. Whenever a fragmentation solution with a better count has been found, we can immediately stop exploration of the current branch of the search tree as soon as the number of cells in the owner fragment exceeds the number of cells of the previously found solution. For sake of simplicity, we leave the details of these two operations out of the pseudocode listing. Unit propagation is however incorporated in Figure 2 and also the cell count is annotated in each node of the search tree. Note that the algorithm in Figure 1 would return the first solution with cell count 8, while a branch-and-bound algorithm involving optimization would return the first minimal solution with cell count 6.

Figure 2 shows the search tree for the following example.

Example 3. The set of dependencies contains two tgds that link treatments with diseases:

$$\mathcal{D} = \{\forall x\, \mathit{Treatment}(x, \mathsf{MedC}) \rightarrow \exists z \mathit{Illness}(x, z),$$

$$\forall x\ \mathit{Treatment}(x, \mathsf{MedA}) \wedge \mathit{Treatment}(x, \mathsf{MedB}) \rightarrow \mathit{Illness}(x, \mathsf{Aids})\}$$

The set of confidentiality constraints states that the disease aids is confidential for any patient and that for patients Pete and Lisa it should not be outsourced that both suffer from the same disease:

$$\begin{aligned}\mathcal{C} = \{&\exists x \mathit{Illness}(x, \mathsf{Aids}),\\ &\exists y(\mathit{Illness}(\mathsf{Pete}, y) \wedge \mathit{Illness}(\mathsf{Lisa}, y))\}\end{aligned}$$

Finally the input instance I is as follows:

I:

Illness	*Name*	*Diagnosis*
	Mary	Aids
	Pete	Flu
	Lisa	Flu

Treatment	*Name*	*Medicine*
	Mary	MedA
	Mary	MedB
	Pete	MedC

The input instance I satisfies all database dependencies. The first candidate set *Inst* finds the following instantiations of confidentiality constraints:

$$\begin{aligned}\mathit{Inst} = \{&\mathit{Illness}(\mathsf{Mary}, \mathsf{Aids}),\\ &\mathit{Illness}(\mathsf{Pete}, \mathsf{Flu}) \wedge \mathit{Illness}(\mathsf{Lisa}, \mathsf{Flu})\}\end{aligned}$$

The unit formula $\mathit{Illness}(\mathsf{Mary}, \mathsf{Aids})$ can be added immediately to the owner fragment. For $\mathit{Illness}(\mathsf{Pete}, \mathsf{Flu})$ we can try both moving it to the owner and the server fragment and hence have two branches in the search tree.

The first fragmentation found with cell count 8 is the following:

F_o:

Illness	*Name*	*Diagnosis*
	Mary	Aids
	Pete	Flu

Treatment	*Name*	*Medicine*
	Mary	MedA
	Pete	MedC

F_s:

Illness	*Name*	*Diagnosis*
	Lisa	Flu

Treatment	*Name*	*Medicine*
	Mary	MedB

The first optimal fragmentation with cell count 6 is the following:

F_o:

Illness	*Name*	*Diagnosis*
	Mary	Aids
	Lisa	Flu

Treatment	*Name*	*Medicine*
	Mary	MedA

F_s:

Illness	*Name*	*Diagnosis*
	Pete	Flu

Treatment	*Name*	*Medicine*
	Mary	MedB
	Pete	MedC

We now briefly analyze the algorithm in terms of correctness and runtime complexity of the proposed algorithm. First of all, the output fragmentation satisfies Definition 4 of horizontal fragmentation correctness:

– Completeness is ensured because when all tuples that have to be moved to the owner fragment have been identified, the remaining tuples of I are moved to the server fragment.

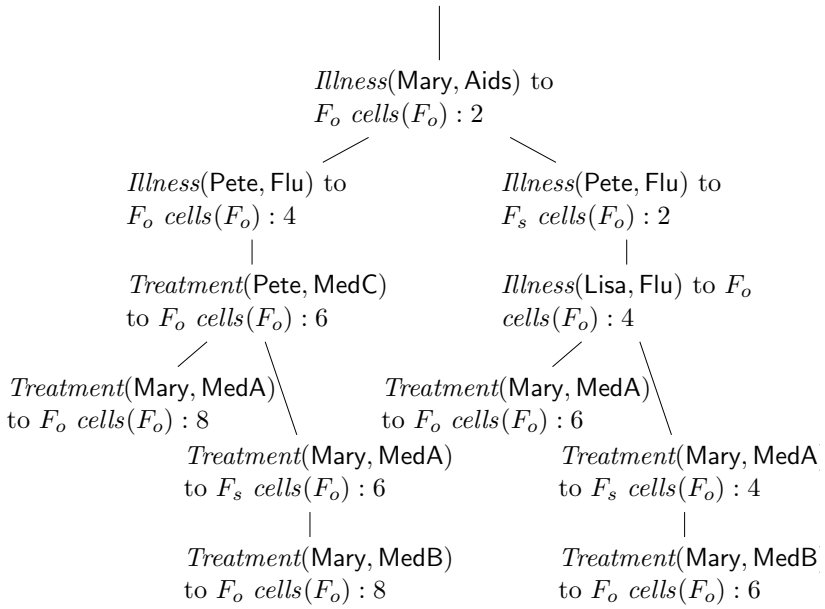

Fig. 2. Example search

- Confidentiality is ensured because on the one hand, all instantiations of confidentiality constraints are handled such that they are not implied by the server fragment. On the other hand the algorithm proceeds such that the server fragment satisfies all database dependencies because no body of a tgd can be fully instantiated whenever the instantiated head does not hold in the server fragment. Hence no deduction of other facts is possible. In terms of fragmentation checking (see Section 4.1), the chase cannot apply any dependencies to F_s.
- Non-redundancy is ensured because ground atoms are contained in I (and hence neither in F_o nor in F_s) before moving them to one of the fragments.

The runtime complexity depends on the number of tuples in the input instance I as follows. Confidentiality constraints (without existential prefix) as well as bodies of tgds are positive conjunctive formulas. Hence their number of instantiations in $I \cup F_s$ is always finite (even with theoretically infinite domains of attribute values) and must indeed be contained in $I \cup F_s$. Consequently, in the worst case every tuple in the input instance I has to be tested whether it has to be moved to the owner or the server fragment. Due to this binary nature, the worst case complexity is exponential in the number of tuples in I. However, average complexity might be a lot better when unit propagation and pruning are applied.

5 Vertical Fragmentation can be Data-Dependent

We now briefly elaborate how vertical fragmentation can be achieved with formula-based confidentiality constraints. In particular, vertical fragmentation can be made data-dependent in the sense that not whole columns are stored in the owner fragment but only sensitive parts of columns. For example, confidentiality of the constraint $\mathit{Illness}(\mathsf{Pete}, y)$ can be achieved by removing only those cells of the *Name* column for which *Name* equals Pete. The remainder of the *Name* column and the *Disease* column can then still be outsourced to the server fragment. Hence, our cell count metrics leads to a better solution in the case that only a part of a column is stored in the owner fragment.

Indeed, this approach yields a form of "hybrid fragmentation": A combination of vertical and horizontal fragmentation can maximize the amount of outsourced data better than each of the techniques alone: If only some values in a column (for example, only some entries in the *Disease* column) must be protected, vertical fragmentation would remove the whole column while horizontal fragmentation only suppresses the rows containing sensitive values. On the other hand, if all values of one column have to be protected (for example, all patient names), vertical fragmentation just removes this column while horizontal fragmentation would have to suppress the whole relation.

The notion of fragmentation checking can also be applied to this hybrid approach: we can handle suppressed cells in the server fragment as null values and apply the chase to the server fragment as in Definition 5; the certain answers can also be computed for open confidentiality constraints and confidentiality is preserved if the answer set is empty. Fragmentation computation has to be modified accordingly such that not the whole row but only some cells of a row are suppressed in the server fragment.

Yet there is a problem if we assume a well-informed and suspicious server. For example, if the server knows the definition of the confidentiality constraint $\mathit{Illness}(\mathsf{Pete}, y)$ then he could suspect that those tuples in the server fragment for which the name is missing actually belong to the patient Pete. This effect is known as "meta-inferences" (see [13]) because although the fragmentation satisfies the formal correctness definition still inference of confidential information is possible on a meta-level. In this case, appropriate countermeasures have to be taken. For example, by moving more name entries to the owner fragment as strictly necessary (and informing the server about it). Or ensuring that the confidentiality constraints lead to a server fragment that satisfies the properties of k-anonymity (see [14]).

6 Related Work and Conclusion

With the introduction of horizontal fragmentation correctness and formula-based confidentiality constraints, we extended the notion of secure fragmentation for data outsourcing (as analyzed in [5, 6] for vertical fragmentation) significantly. On the one hand we showed that horizontal fragmentation gives rise to a new

application of the chase for the fragmentation checking problem (as used in [8, 10, 12, 11] for similar purposes). On the other hand we presented an algorithm that computes a correct horizontal fragmentation and at the same time can be used to optimize the fragmentation with respect to some criteria like for example our cell count criterion; other such criteria can also be used with the algorithm.

Open questions remain: other fragments of first-order logic can be studied for database dependencies and confidentiality constraints; further research could investigate the behavior and performance of horizontal fragmentation when the user queries or updates his outsourced data; some query strategies are already analyzed for vertical fragmentation in [3, 6]. Moreover the area of hybrid fragmentation can be advanced and the problem of meta-inferences can be investigated further. An in-depth analysis of applications of k-anonymity techniques [14] to data outsourcing is also desirable.

References

1. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: SIGMOD Conference, ACM (2002) 216–227
2. Hacigümüs, H., Iyer, B.R., Mehrotra, S.: Query optimization in encrypted database systems. In: DASFAA 2005. Volume 3453 of Lecture Notes in Computer Science., Springer (2005) 43–55
3. 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, CIDR 2005. (2005) 186–199
4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. In: ESORICS 2007. Volume 4734 of LNCS., Springer (2007) 171–186
5. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a few: Outsourcing data while maintaining confidentiality. In: ESORICS 2009. Volume 5789 of LNCS., Springer (2009) 440–455
6. 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: DBSec. Volume 5645 of LNCS., Springer (2009) 225–239
7. Biskup, J., Tsatedem, C., Wiese, L.: Secure mediation of join queries by processing ciphertexts. In: ICDE Workshops, IEEE Computer Society (2007) 715–724
8. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science **336**(1) (2005) 89–124
9. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Transactions on Database Systems **4**(4) (1979) 455–469
10. Afrati, F.N., Kolaitis, P.G.: Repair checking in inconsistent databases: algorithms and complexity. In: 12th International Conference on Database Theory, ICDT. Volume 361 of ACM International Conference Proceeding Series., ACM (2009) 31–41
11. Brodsky, A., Farkas, C., Jajodia, S.: Secure databases: Constraints, inference channels, and monitoring disclosures. IEEE Transactions on Knowledge & Data Engineering **12**(6) (2000) 900–919

12. Stouppa, P., Studer, T.: A formal model of data privacy. In: Ershov Memorial Conference 2006. Volume 4378 of LNCS., Springer (2007) 400–408
13. Biskup, J., Gogolin, C., Seiler, J., Weibert, T.: Requirements and protocols for inference-proof interactions in information systems. In: ESORICS 2009. Volume 5789 of LNCS., Springer (2009) 285–302
14. Ciriani, V., di Vimercati, S.D.C., Foresti, S., Samarati, P.: k-anonymity. In: Secure Data Management in Decentralized Systems. Volume 33 of Advances in Information Security. Springer (2007) 323–353