Keeping Secrets by Separation of Duties while Minimizing the Amount of Cloud Servers

Ferdinand Bollwein[1] and Lena Wiese[2]

[1] Institute of Computer Science, TU Clausthal
ferdinand.bollwein@tu-clausthal.de
[2] Institute of Computer Science, University of Goettingen
wiese@cs.uni-goettingen.de

Abstract. In this paper we address the problem of data confidentiality when outsourcing data to cloud service providers. In our separation of duties approach, the original data set is fragmented into insensitive subsets such that each subset can be managed by an independent cloud provider. Security policies are expressed as sets of confidentiality constraints that induce the fragmentation process. We assume that the different cloud providers do not communicate with each other so that only the actual data owner is able to link the subsets and reconstruct the original data set. While confidentiality is a hard constraint that has to be satisfied in our approach, we consider two further optimization goals (the minimization of the amount of cloud providers and the maximization of utility as defined by visibility constraints) as well as data dependencies that might lead to unwanted disclosure of data. We extend prior work by formally defining the confidentiality and optimization requirements as an optimization problem. We provide an integer linear program (ILP) formulation and analyze different settings of the problem. We present a prototype that exploits a distributed installation of several PostgreSQL database systems; we give an in-depth account of the sophisticated distributed query management that is enforced by defining views for the outsourced data sets and rewriting queries according to the fragments.

1 Introduction

Data outsourcing and data processing in cloud services now-a-days manifest in different variants and for different use cases: numerous providers offer cloud services where data is processed off-premise and is no longer under the control of the actual data owner. Using such cloud services offers several advantages like:

- scalability: customers can book resources according to their demand leading to a reduction of hardware and maintenance costs;
- flexibility: cloud servers can run with an optimal payload and hence the overall consumption of energy can be reduced;
- availability: customers can access their data from anywhere – independent of their physical location;

– reliability: cloud providers offer service level agreements and invest in reliability of their systems so that data loss is reduced.

In particular for large data sets or when data can be accessed by many different parties, using cloud services for data outsourcing offers many benefits. For the purposes of this article we categorize data outsourcing into three different variants:

– **Data storage** (single-owner read and write access): a single data owner manages his or her data remotely at outsourcing providers; the data owner has read and write access on the data. As one example for this setting, businesses can profit by sparing the cost of maintaining an own computing center; as another example, a patient can maintain his or her personal electronic health record in the cloud.
– **Data publishing** (single-owner write, multiple-user read access): a single data owner stores some data at outsourcing providers. Other users can then query the data (that is, with read-only access). Access control may be enforced on the published data: certain data items may only be queried by privileged users. A particular variant of data publishing is the statistical evaluation of the outsourced data: in this case, the data can even be distorted as long as the final evaluation on the distorted data does not diverge much from the evaluation on the original data.
– **Data sharing** (multiple-owner read and write access): multiple data owners manage their data collaboratively where each data owner may selectively be entitled only to certain read and write accesses. In the business example, in contrast to data storage, different business units may share data and have the added value of executing fine-grained access control; in the electronic health record example, patients can selectively allow read and write access to some of their health data for medical personnel.

However, outsourcing data also means that the user loses control over the data and the cloud providers have to be trusted in order to ensure confidentiality of sensitive or business-critical data. The three outsourcing variants each have different security requirements which we now briefly discuss.

When using the *data storage* variant, data should be inaccessible to other users or the cloud service provider. One possibility to ensure confidentiality would be to encrypt the data stored in the cloud database, however, this limits the provider's ability of processing the data to answer complex queries by the user. To achieve query answering on ciphertexts, property-preserving encryption schemes are available that allow to sort ciphertexts, or to search for encrypted keywords on the ciphertexts. While these schemes enable certain operations on the encrypted data they come at the cost of an increased overhead; prototype systems that apply such encryption schemes are [27, 31] for SQL databases (which also feature homomorphic encryption for aggregations) and [34] for NoSQL databases.

When using the *data publishing* variant, only certain characteristics of the data are confidential. These characteristics can be hidden by distorting the

data with some noise. Examples for the publishing of distorted data include k-anonymity [28] and differential privacy [20]. For the example use cases, business-critical or medical data would only published in a curated version including slightly modified or generalized values.

In case of the *data sharing* variant, data should selectively be accessible to some users but should be hidden from other users or third parties. For the purpose of encrypted data sharing with multiple owners, *multi-user* property-preserving encryption schemes exist but the distribution of appropriate cryptographic keys is a major complication.

As an alternative, this article proposes a *separation of duties* approach to address the complexity of encryption – however, we want to reinforce that in a real-world system a combination of separation of duties and encryption would be of added practical value. Our proposed approach applies to the data outsourcing variant of data storage, where data confidentiality is achieved by distributing data fragments among different separate cloud service providers – with a specific focus on data storage in cloud databases ("database-as-a-service"). If in addition selective access rights on the distributed fragments are given to several users, the data outsourcing variant of data sharing can also profit from our separation of duties approach. We refrain however from distorting data as would be necessary for some data publishing scenarios mentioned above. A particular scenario that our approach is suited for is the outsourcing of medical data into repositories. Here several data providers can submit their data into these repositories. The important difference to encryption-based approaches is that statistical evaluations are still possible on plaintext data; support for statistics on certain attribute combinations can be enforced by visibility constraints.

Generally, the term separation of duties means that a specific task is handled by multiple entities to prevent malicious behavior which could be carried out by a single entity in control of the whole task. In the context of preserving confidentiality in cloud databases, this is based on the observation that in many scenarios data only becomes sensitive in association with other data. By distributing the data among multiple database servers with a technique called *vertical fragmentation*, these sensitive associations can be broken up such that each server only maintains an insensitive portion of the data. The fragmentation process is guided by a security policies that contains so-called confidentiality constraints. As long as the servers are not collaborating to reestablish the association, this separation of duties approach ensures the confidentiality of the underlying data. The huge advantage of our approach is that the data can be outsourced in plaintext because and the data do not have to be encrypted which would drastically limit the servers' ability to process client queries. Moreover, no noise is added and the data outsourced plaintext data retain their original values.

It is a challenging task to decide how the data is distributed among the servers. On the one hand, the security requirements have to be met but on the other hand the number of involved servers should be relatively small to limit the costs for the user and ensure efficient querying. Therefore, the proposed separation of duties approach can be viewed as a typical mathematical optimi-

zation problem. In this article we consolidate and extend our prior work [6, 7] in a multi-relational environment. Moreover, we extend the benchmarking of our prototype implementation which is capable of distributing data (more precisely, vertical fragments) among several database servers and which appropriately analyzes and rewrites arbitrary SQL queries. In detail, in this article we make the following contributions:

- We investigate separation of duties in a realistic data model; in particular, we consider multiple database relations as well as dependencies between data which are allowed to span multiple relations – both of these are important in real-world database systems to achieve non-redundancy and to avoid anomalies for example by database normalization.
- We formulate separation of duties as a mathematical optimization problem according to several optimization goals – satisfying confidentiality constraints under dependencies while minimizing the amount of database servers, maximizing the amount of satisfied visibility constraints.
- We provide article a benchmarking of the optimization procedure (using IBM CPLEX) on the widely used TPC-E data schema.
- We give a detailed account of the query rewriting approach followed when accessing data that are distributed among different fragments.
- We provide details of a benchmarking of distributed query execution on the fragmented TPC-H dataset. In particular, our approach can execute all of the TPC-H benchmark queries – while currently existing approaches using property-preserving encryption are unable to execute the entire query set.

We start this article with a survey of related work in Section 2. Section 3 sets the necessary terminology; Section 4 analyzes the theory of several Separation of Duties problems; Section 5 provides a translation into an integer linear program; Sections 6 and 7 describe the implementation and evaluation; Section 8 concludes the article.

2 Related Work

Security and privacy have been challenging tasks ever since the introduction of the principle of data outsourcing and a lot of research has been carried out to address different aspects of this problem such as access control, data confidentiality and data integrity. Security can be achieved by encryption – carried out by the user before outsourcing the data to the database. Still, encryption operations are generally very costly and moreover, existing cryptographic techniques can still not be efficiently used to evaluate more complex queries like, for instance, computations on the data. In this paper we focus on data confidentiality by data fragmentation and distribution without encryption; however our proposed approach can indeed be combined with conventional encryption (as in [1]) or even novel encryption approaches (that we also applied in prior work as for example order-preserving encryption [32], searchable encryption [33] or even fuzzy searchable encryption [21, 23]).

Vertical fragmentation approaches split a database into two or more fragments consisting each of a subset of the attributes (that is, columns) of the entire database. The theory of vertical fragmentation for relational database systems is well-studied. As an early resource, [9] study vertical fragmentation that also takes transaction processing costs into account. Fragmentation is also extensively covered in the standard textbook [26]. A more recent comparative evaluation of vertical fragmentation approaches is provided in [25]; however all of these approaches do not consider fragmentation as a security mechanism.

Vertical fragmentation can be used to break sensitive associations between attributes. Vertical fragmentation for data outsourcing was first analyzed from a security point of view in [1] where a single relation is divided into *two* fragments. They model sensitive associations between columns of the single relation as subsets of its columns. In each fragment some values of a tuple are stored as plaintext while each fragment always additionally contains each entire tuple in encoded form (for example, encrypted); that is, they rely on encryption whenever it is impossible to meet the confidentiality requirements with those two servers only. The two fragments are outsourced to two distinct *honest-but-curious*, *non-communicating* servers. A user runs a client software for database management, query optimization and query postprocessing. Query execution performance is good for queries covering plaintext values but worse for querying encoded data. The authors provide strategies to optimize query execution plans and identify an optimal vertical fragmentation according to some cost matrix. In contrast to this work, our approach supports more than two external servers as well as a trusted owner server, so that we can refrain from using any form of encryption and we consider more optimization goals.

Several approaches extended and improved this first analysis. [12] and [14] pioneer the idea of a single data owner safeguarding as few data as possible of a single relation. To do so, a database table is split vertically into a *server fragment* (to be stored at an untrusted cloud server) and an *owner fragment* (to be stored at a trusted local database server). Again, only two fragments are considered. The aim is to store a minimal subset of columns in the local database such that the remaining set of columns which is stored at the untrusted server is insensitive. This approach works totally without encryption because the owner site is assumed to be trusted. The authors define fragmentation correctness with respect to completeness, confidentiality and non-redundancy, and analyze fragmentation metrics to assess quality of a fragmentation. A complexity analysis is based on the weighted minimum target hitting set problem.

Other proposals such as [10, 15, 13, 11, 19, 17, 18] use vertical fragmentation as a tool to split sensitive associations in a database table. It is required that those fragments do not have an attribute in common such that the fragments are *unlinkable*. Due to the unlinkability, the fragments can possibly be stored at a single untrusted server which is then in possession of all the data but cannot establish sensitive associations. In particular, [19] introduces the concept of k-loose associations: while associations between single tuples (or the confidential values therein) in two fragments remain protected, more general associations between

groups (of size k) of confidential values can be published and hence improve data visibility. For a detailed security analysis, probabilities of one-to-one associations between values given the published fragments and k-loose associations are analyzed. The authors consider both confidentiality and visibility constraints but assume that there is no conflict between these constraints.

In [18] the concept of *data dependencies* is introduced. The authors state that certain combinations of attributes can be used by a sophisticated untrusted server to draw conclusion about other attributes which could potentially lead to the exposure of sensitive data. More specifically, [4, 2] consider certain classes of data dependencies (in particular, subcases of equality generating dependencies and tuple generating dependencies) and investigate their impact on information disclosure.

[8] proposes vertical fragmentation into two fragments: one with confidential and one with non-confidential attributes; only confidential attributes are encrypted with the Advanced Encryption Standard (AES). Associations between attributes are not considered. An analysis reveals that joins between the confidential and non-confidential attributes are costly.

We extend and consolidate these prior approaches by supporting multiple relations while combining several optimization goals into one problem definition.

In addition, some approaches go beyond vertical fragmentation and consider other kinds of fragmentations: Expressive constraints and dependencies in first-order logic have previously been analyzed in [4] for vertical as well as in [35] for horizontal confidentiality-preserving fragmentations.

Several approaches cover only data publishing – as opposed to data storage or data sharing as we do. In particular, [37, 39] studies data publishing and hierarchical data partitioning under the notion of differential privacy which is a probabilistic measure of data confidentiality. Roughly, the probability distribution of the published data should not diverge substantially from similar data sets. To achieve this they partition the original datasets into subsets and introduce noise to ensure confidentiality for certain statistical queries; these queries are however not general enough (mostly count queries are considered). As another approach for data publishing is [24] that enumerates all possible hybrid fragmentations (what the authors call "hierarchical partitioning") of a table. Afterwards, an optimal fragmentation is anonymized by generalizing data in each fragment with a new value. More recently, [38] combine vertical fragmentation with k-anonymity, l-diversity, and t-closeness to achieve more efficiency when publishing multi-dimensional data. In contrast to these (as well as related) approaches, we focus on data storage and data sharing instead of data publishing – hence in our approach data are not distorted.

3 Background

As the underlying data model, we focus on the formal definition of a relational database. As usual, a relation schema $R(\{a_1, \ldots, a_n\})$, or simply $R(a_1, \ldots, a_n)$, consists of a *relation name* R and a finite set of *attributes* $\{a_1, \ldots, a_n\}$ with

$n \geq 1$. Each attribute a_i is associated with a specific domain which is denoted by the expression $\mathrm{dom}(a_i)$. Next, a *relation* r, also denoted by $r(R)$, over the relation schema $R(a_1, \ldots, a_n)$ is defined as an ordered set of *n-tuples* $r = (t_1, \ldots, t_m)$ such that each *tuple* r_j is an ordered list $t_j = v_1, \ldots, v_n$ of values $v_i \in \mathrm{dom}(a_i)$ or $v_i = \mathsf{NULL}$. The *degree* of a relation is defined as the number of attributes in r.

A *database schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ is defined by a name D and a set of relation schemes $R_i(A_i)$ where A_i denotes the corresponding set of attributes. Finally, a *database state* $d = \{r_1, \ldots, r_N\}$ over a database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ is a set of relations such that each r_i is a relation over the respective relation schema $R_i(A_i)$.

As a tuple identifier tid_s for relation schema R_s a subset of A_s is chosen that is a candidate key: the tuple values are functionally dependent on the key; that is, two tuples with the same identifier value would be identical on all other values. Formally, if r_s is a relation over the schema $R_s(A_s)$ and $\mathrm{tid}_s \subseteq A_s$, for the two tuples $t_1, t_2 \in r_s$ the following holds

$$t_1[\mathrm{tid}_s] = t_2[\mathrm{tid}_s] \Rightarrow t_1 = t_2$$

The set of all tuple identifiers is denoted by $\mathrm{tid} := \bigcup_{s=1}^{N} \mathrm{tid}_s$.

To illustrate the individual steps (namely, setup, select, insert, delete and update), a small database D consisting of two tables **D** and **P** in a hospital scenario serves as a running example. The first table stores information about doctors and the second table stores information about patients:

$$\begin{aligned} D = \{ \quad & \mathbf{D}(\mathsf{DocID}, \mathsf{Name}, \mathsf{DoB}, \mathsf{ZIP}, \mathsf{Specialty}), \\ & \mathbf{P}(\mathsf{PatID}, \mathsf{Name}, \mathsf{DoB}, \mathsf{ZIP}, \mathsf{Diagnosis}, \mathsf{Treatment}, \mathsf{DocID}) \ \} \end{aligned}$$

where DocID and PatID serve as tuple identifiers.

4 Separation of Duties Problems

Analogous to [1] and [12] we assume that Cloud service providers are "honest but curious". This means that servers handle requests and answer queries correctly; but, while they do not manipulate the stored or returned data, still they analyze data and user behavior and try to gain sensitive information from it.

A security policy consisting of confidentiality constraints (see Definition 1) describes what information is confidential in terms of subsets of attributes (that is, column names) of relations. The presented separation of duties approach aims at protecting confidentiality of either all values in an individual column (the so-called singleton constraints) or the combination of values for the same tuple in different columns (the so-called association constraints). As an extension to prior work, in a database containing multiple relations, sensitive associations can exist among relations. This is expressed in the following definition of multi-relational confidentiality constraints (Definition 1).

Definition 1. ***(Multi-Relational Confidentiality Constraints)*** *Let the sets $A_1, \ldots, A_N$ denote sets of attributes that are pairwise disjoint; furthermore, let $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ be a database schema and $d = \{r_1, \ldots, r_N\}$ a database state over D. A* multi-relational confidentiality constraint *on D is defined by a subset of attributes $c \subseteq \bigcup_{s=1}^{N} A_s$. A multi-relational confidentiality constraint c with $|c| = 1$ is called a* singleton constraint*. If $|c| > 1$ it is called an* association constraint*.*

The condition that the set of attributes are pairwise disjoint is introduced to assure that attributes can uniquely be associated with the according relation schema. This can easily be achieved by choosing a suitable naming convention (like prependeding relation names to the attributes).
Vertical fragmentation is used to meet the security requirements expressed by the confidentiality constraints. In Definition 2, correctness of vertical fragmentation is defined based on the three properties of completeness, disjointness and reconstruction.

Definition 2. ***(Multi-Relational Vertical Fragmentation, cardinality of a fragmentation, physical fragments)*** *Let $A_1, \ldots, A_N$ denote sets of attributes that are pairwise disjoint; furthermore, let $d = \{r_1, \ldots, r_N\}$ be a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A set of fragments $\mathbf{f} = (f_0, \ldots, f_k)$ where $f_j \subseteq \bigcup_{s=1}^{N} A_s$ for all f_j is called a* correct (multi-relational) vertical fragmentation *of d if the following conditions are met:*

- ***Completeness:*** $\bigcup_{i=0}^{k} f_j = \bigcup_{s=1}^{N} A_s$
- ***Disjointness:*** $f_i \cap f_j \subseteq tid, \forall\, f_i \neq f_j$ *with* $f_i, f_j \neq \emptyset$
- ***Reconstruction****:* $tid_s \subset (f_j \cap A_s),$ *if* $f_j \cap A_s \neq \emptyset$

A fragmentation that satisfies all properties except the disjointness is called a lossless *(multi-relational) vertical fragmentation of r. The* cardinality *of a vertical fragmentation* $\mathrm{Card}(\mathbf{f})$ *is defined as the number of nonempty fragments in* $\mathbf{f}$*:* $\mathrm{Card}(\mathbf{f}) = \sum_{\substack{j=0 \\ f_j \neq \emptyset}}^{k} 1$.
For fragment f_j, its multirelational physical fragment *is the set of projections*

$$d_j := \{\pi_{f_j \cap A_1}(r_1), \ldots, \pi_{f_j \cap A_N}(r_N)\}$$

which is a database over the database schema $D_j = \{R_1(f_j \cap A_1), \ldots, R_N(f_j \cap A_N)\}$. The individual relations $\pi_{f_j \cap A_s}(r_s)$ for $s \in \{1, \ldots, N\}$ are called relation fragments*.*

Note that each fragment f_j contains a subset of the attributes of each original table; the physical fragment d_j contains subtables of all the original tables. The cardinality of the fragmentation $\mathbf{f}$ denotes the amount of non-empty fragments f_j. We will later on minimize the cardinality in order to minimize the amount of occupied external cloud servers.

In this definition the completeness property ensures that every attribute is contained in at least one fragment. The disjointness property prevents unnecessary copies of attributes that are not tuple identifiers; in other words, only

tuple identifiers are allowed to be contained in more than one fragment. The reconstruction property makes sure that a fragment contains all necessary tuple identifiers to reconstruct the original relations by joining the corresponding relation fragments. More precisely, the reconstruction property ensures that a tuple identifier tid_s is contained in a fragment f_j, if and only if f_j also contains a non-tuple identifier attribute from relation r_s. On the one hand, this ensures that the individual relations can be reconstructed using join operation on the tuple identifiers and on the other hand prevents fragments that only contain the tuple identifier but no non-tuple identifier attribute of a specific relation.

In the following subsections we discuss three variants of separation of duties by vertical fragmentation.

4.1 Standard Separation of Duties

As a minimum, confidentiality-preserving fragmentations must obey security policies by ensuring that not all attributes contained in a confidentiality constraint are part of a fragment at the same time. The only exception is the owner fragment f_0 that is stored on the trusted client side and contains all singleton constraints as well as subsets of association constraints if they cannot be satisfied by distributing their attributes among several server fragments.

Definition 3. ***(Confidentiality)*** *Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes. Given a database state $d = \{r_1, \ldots, r_N\}$ over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ and a set of confidentiality constraints C. A vertical fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ is* confidentiality-preserving *with respect to a set of confidentiality constraints C if the following condition is met:*

$$c \not\subseteq f_j \text{ for all } c \in C \text{ and } j \in \{1, \ldots, k\}$$

In this definition f_0 is the owner fragment (to be stored at the trusted database server) and $f_1, \ldots, f_k$ are the k server fragments (to be stored at the k untrusted database servers). A confidentiality-preserving vertical fragmentation therefore requires that the combination of attributes defined by a confidentiality constraint is *not* jointly visible in a server fragment. For singleton constraints, this implies that the corresponding attribute must be placed in the owner fragment.

To avoid redundancy and unwanted interactions with the tuple identifiers, we impose some restrictions on the set of confidentiality constraints. In particular, tuple identifiers are assumed to be unsensitive information – because they are needed to reconstruct the original relations – and hence should not be contained in confidentiality constraints.

Definition 4. ***(Well-defined Confidentiality Constraints)*** *Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes and let $\text{tid}_s \subset A_s$ denote the designated tuple identifier for $A_1, \ldots, A_N$ respectively. Moreover, let $d = \{r_1, \ldots, r_N\}$ denote a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A set of confidentiality constraints C is* well-defined *if it satisfies the following conditions:*

- $c \not\subseteq c'$ *for all* $c, c' \in C$ *with* $c \neq c'$
- $c \cap tid_s = \emptyset$ *for all* $c \in C$ *and* $s \in \{1, \ldots, N\}$ *with* $c \subseteq A_s$

The first condition requires that no confidentiality constraint c is a subset of another c' – due to the requirements for a confidentiality-preserving multi-relational vertical fragmentation, the restriction that c is not jointly visible in any server fragment already implies that c' is not jointly visible in any server fragment.

The second condition of the previous definition further implies that confidentiality constraints c that contain only attributes from a single relation are not allowed to contain tuple identifier attributes if they contain at least one non-tuple identifier attribute. Such constraints can simply be replaced by the semantically equivalent constraints $c \setminus \mathrm{tid}_s$. This will avoid unnecessary case differentiations in the remainder of this work.

Continuing our example, confidentiality constraints can express that patients' and doctors' names are highly confidential and that (as a kind of quasi-identifiers) the combinations of a patient's DoB, ZIP code and diagnosis as well as a doctor's DoB and ZIP code must not be revealed together.

$$C = \{\{\mathbf{P}.\mathsf{Name}\}, \{\mathbf{D}.\mathsf{Name}\}, \{\mathbf{P}.\mathsf{DoB}, \mathbf{P}.\mathsf{ZIP}, \mathbf{P}.\mathsf{Diagnosis}\}, \{\mathbf{D}.\mathsf{DoB}, \mathbf{D}.\mathsf{ZIP}\}\}$$

A confidentiality-preserving fragmentation consists of one owner fragment

$$f_0 = \{\mathbf{P}_0(\mathsf{PatID}, \mathsf{Name}), \mathbf{D}_0(\mathsf{DocID}, \mathsf{Name})\}$$

and two server fragments

$$f_1 = \{\mathbf{P}_1(\mathsf{PatID}, \mathsf{DoB}, \mathsf{DocID}), \mathbf{D}_1(\mathsf{DocID}, \mathsf{ZIP})\}$$

and

$$f_2 = \{\mathbf{P}_2(\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Diagnosis}, \mathsf{Treatment}), \mathbf{D}_2(\mathsf{DocID}, \mathsf{DoB}, \mathsf{Specialty})\}.$$

It can be seen that no server fragment contains the entire set of attributes specified in one confidentiality constraint. Moreover, the tuple identifiers are the necessary information that enables the owner to reconstruct the two original tables. Again note that each fragment contains a subset of the attributes of each of the two original relations $\mathbf{P}$ and $\mathbf{D}$; the cardinality of our fragmentation is 3 because we obtained one owner fragment and two server fragments.

As a last component, we consider storage space capacities for the servers. We specify a weight function that assigns a weight to each set of attributes that denotes the capacity consumption of the set: $w_d : \mathcal{P}(A) \longrightarrow \mathbb{R}^+$. A simple weight function could for example count the number of attributes in the set. We then consider a maximum capacity W_j for each server S_j and require that the summed weights of the fragment do not exceed the capacity of the server that hosts this fragment.

With these preliminaries the definition of the Standard Multi-relational Separation of Duties Problem considering the cardinality of the fragmentation (as in Definition 2) as well as the confidentiality (as in Definition 3) is as follows:

Definition 5. ***(Multi-relational Separation of Duties)*** *Given a database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, a database state $d = \{r_1, \ldots, r_N\}$, a well-defined set of multi-relational confidentiality constraints C, tuple identifiers $tid_s \subset A_s$ for all $s \in \{1, \ldots, N\}$, a weight function w_d, servers $S_0, \ldots, S_k$ and corresponding maximum capacities $W_0, \ldots, W_k \in \mathbb{R}_0^+$, the Multi-relational Separation of Duties Problem consists of finding a correct confidentiality-preserving fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ of minimal cardinality* $\mathrm{Card}(\mathbf{f})$ *such that the capacities of the storage are not exceeded, i.e. $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$.*

The maximum capacity W_0 of the owner fragment can be set such that the owner fragment only stores the attributes in the singleton constraints – which cannot be outsourced due to their sensitivity. This can be achieved by choosing a suitable capacity of the owner fragment. Yet, it must be considered that the correct tuple-identifier attributes must also be part of the owner fragment to satisfy the reconstruction property.

Lemma 1. *If the set of singleton constraints $A^* := \{c \in C \mid |c| = 1\}$ denotes the set of all sensitive attributes, when setting W_0 to*

$$W_0 = \sum_{s:A_s \cap A^* \neq \emptyset} w_d(tid_s) + \sum_{c \in C:|c|=1} w_d(c).$$

the owner fragment only stores the attributes contained in singleton constraints and the necessary tuple identifiers.

The Standard Separation of Duties Problem can be viewed as a combination of two famous NP-hard problems, the bin packing problem due to the capacity constraints of the storage locations and the vertex coloring problem due to the confidentiality constraints.

4.2 Visibility Constraints

In the multi-relational scenario it is very important to control the resulting fragmentation in order to increase the utility of the fragmented database and avoid unnecessary joins when executing queries on the distributed fragments. Increased usability means that certain combinations of attributes are stored on a single server because they are often queried together. The notion of visibility constraints will be adapted to this scenario: visibility constraints are defined as subsets of attributes that should be placed in a single fragment – in this case, we say that the visibility constraint is satisfied. Satisfaction of visibility should only be enforced if the resulting fragmentation is not in conflict with the confidentiality requirements. That is, confidentiality requirements are ranked higher than visibility requirements. Formally, the definition of visibility constraints and the amount of satisfied visibility constraints is as follows:

Definition 6. ***(Visibility constraint, satisfaction)*** *Let $d = \{r_1, \ldots, r_N\}$ be a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$.*

A (multi-relational) visibility constraint *over D is a subset of attributes $v \subseteq \bigcup_{s=1}^{N} A_s$. A multi-relational fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ satisfies v if there is a $0 \leq j \leq k$ such that $v \subseteq f_j$. If such an f_j exists, define* $\mathrm{Sat}_v(\mathbf{f}) \coloneqq 1$. *Otherwise, define* $\mathrm{Sat}_v(\mathbf{f}) \coloneqq 0$. *For any set V of visibility constraints, the number of satisfied visibility constraints is*

$$\mathrm{Sat}_V(\mathbf{f}) \coloneqq \sum_{v \in V} \mathrm{Sat}_v(\mathbf{f})$$

Note that as opposed to other approaches we treat visibility constraints as soft constraints; that is, conflicts in the specification are allowed and visibility constraints will be satisfied only if the confidentiality can still be ensured. Hence, it may happen that not all visibility constraints can be satisfied.

For example, focusing on our patient example table, in order to preserve the privacy of the patients, the confidentiality constraint $c = \{\mathsf{DoB}, \mathsf{ZIP}\}$ is enforced; in addition, a visibility constraint $v = \{\mathsf{ZIP}, \mathsf{Diagnosis}\}$ is introduced that enables the statistical evaluation of the frequency of illnesses per ZIP code. Hence, one possible privacy-preserving fragmentation is given by

$$\mathbf{f} = \{f_0, f_1, f_2, f_3\}$$

with:

$$f_0 = \emptyset, \qquad f_1 = \{\mathsf{PatID}, \mathsf{DoB}\},$$
$$f_2 = \{\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Treatment}\}, \; f_3 = \{\mathsf{PatID}, \mathsf{Diagnosis}\}.$$

Another privacy-preserving fragmentation is given by

$$\mathbf{f}' = \{f'_0, f'_1, f'_2, f'_3\}$$

with:

$$f'_0 = \emptyset, \qquad f'_1 = \{\mathsf{PatID}, \mathsf{DoB}\},$$
$$f'_2 = \{\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Diagnosis}\}, \; f'_3 = \{\mathsf{PatID}, \mathsf{Treatment}\}.$$

The important thing to notice here is that both fragmentations satisfy the confidentiality constraint but in $\mathbf{f}$ the attributes in v are spread among two servers while in $\mathbf{f}'$ they are on the same server; more formally, $\mathrm{Sat}_v(\mathbf{f}) = 0$ while $\mathrm{Sat}_v(\mathbf{f}') = 1$. As a result, the second fragmentation $\mathbf{f}'$ is better because a query for the two attributes ZIP and Diagnosis can be answered by a single server (the one hosting f'_2) without the need to join on patient ID.

Minimizing the number of servers versus maximizing the number of fulfilled visibility constraints are two contrary goals. That is why in the following definition of the Extended Multi-relational Separation of Duties problem we introduce a weighted sum of these two goals using two weights α_1 and α_2. Note that satisfying the confidentiality constraints is still a hard constraint and will be mandatory. Moreover, omitting the disjointness property of fragmentation helps increase the number of fulfilled visibility constraints. Therefore, in the following problem statement, only a *lossless* fragmentation is required.

Definition 7. ***(Extended Multi-relational Separation of Duties)*** *Given database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, a database state $d = \{r_1, \ldots, r_N\}$, a set of well-defined multi-relational confidentiality constraints C, visibility constraints V, tuple identifiers $tid_s \subseteq A_s$ for all $s \in \{1, \ldots, N\}$, a weight function w_d, servers $S_0, \ldots, S_k$ with maximum capacities $W_0, \ldots, W_k \in \mathbb{R}_0^+$ and positive weights $\alpha_1, \alpha_2 \in \mathbb{R}_0^+$, find a lossless confidentiality-preserving fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ of minimal cardinality which satisfies $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$ such that the weighted sum $\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f})$ is minimal.*

Using the weighted sum serves the following two purposes:

1. $\alpha_1 \operatorname{Card}(\mathbf{f})$ is responsible for minimizing the cardinality (amount of fragments) of the fragmentation. Hence we aim to use as few external servers as possible to store the server fragments.
2. By subtracting $\alpha_2 \operatorname{Sat}_V(\mathbf{f})$ each satisfied visibility constraint will lower the overall objective. Hence we aim to maximize the amount of satisfied visibility constraints.

The obvious question arises of how to appropriately choose the weights α_1 and α_2. In the following lemma we show that choosing the weights such that $\alpha_2|V| < \alpha_1$ results in assigning highest priority to the minimization of the cardinality of the fragmentation. Among those cardinality-minimizing fragmentations the number of satisfied visibility constraints should be maximal.

Lemma 2. *Consider weights $\alpha_1 > 0$ and $\alpha_2 > 0$ satisfying $\alpha_2|V| < \alpha_1$. If $\mathbf{f}$ is a solution to the Extended Multi-Relational Single-Relational Separation of Duties Problem and $\mathbf{f}'$ is a lossless confidentiality-preserving fragmentation that does not violate the capacity constraints $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$, the following statements hold:*

1. $\operatorname{Card}(\mathbf{f}') \geq \operatorname{Card}(\mathbf{f})$
2. *If* $\operatorname{Card}(\mathbf{f}') = \operatorname{Card}(\mathbf{f})$, *then* $\operatorname{Sat}_V(\mathbf{f}) \geq \operatorname{Sat}_V(\mathbf{f}')$

Proof. Let $\mathbf{f}$, $\mathbf{f}'$ and α_1 and α_2 be as stated in the lemma. First, Statement 1 is proven by contradiction: suppose $\operatorname{Card}(\mathbf{f}') < \operatorname{Card}(\mathbf{f})$ which is equivalent to

$$\operatorname{Card}(\mathbf{f}) - \operatorname{Card}(\mathbf{f}') \geq 1 \tag{1}$$

because both $\operatorname{Card}(\mathbf{f})$ and $\operatorname{Card}(\mathbf{f}')$ are positive integer values. Furthermore, because $\mathbf{f}$ is a solution to the Extended Multi-Relational Separation of Duties Problem, the following inequality holds:

$$\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f}) \leq \alpha_1 \operatorname{Card}(\mathbf{f}') - \alpha_2 \operatorname{Sat}_V(\mathbf{f}') \tag{2}$$

At most $|V|$ visibility constraints can be satisfied, such that $0 \leq \alpha_2 \operatorname{Sat}_V(\mathbf{f}) \leq \alpha_2|V|$. Thus, because $\alpha_2 \operatorname{Sat}_V(\mathbf{f}') \geq 0$ the following inequality can be derived:

$$\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_1 \operatorname{Card}(\mathbf{f}') \leq \alpha_2 \operatorname{Sat}_V(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f}') \leq \alpha_2 \operatorname{Sat}_V(\mathbf{f}) \leq \alpha_2|V| \tag{3}$$

Together, Inequality 3 and the assumption that $\alpha_2|V| < \alpha_1$ in the lemma lead to the following inequality:

$$\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_1 \operatorname{Card}(\mathbf{f}') < \alpha_1 \tag{4}$$

As α_1 is assumed to be greater than zero, the inequality

$$\operatorname{Card}(\mathbf{f}) - \operatorname{Card}(\mathbf{f}') < 1 \tag{5}$$

must be satisfied which contradicts Inequality 1.

Up next, Statement 2 is proven by contradiction. Hence, it is assumed that $\operatorname{Card}(\mathbf{f}') = \operatorname{Card}(\mathbf{f})$ and $\operatorname{Sat}_V(\mathbf{f}) < \operatorname{Sat}_V(\mathbf{f}')$.

The inequality

$$\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f}) \leq \alpha_1 \operatorname{Card}(\mathbf{f}') - \alpha_2 \operatorname{Sat}_V(\mathbf{f}') \tag{6}$$

as above can now be simplified to

$$\alpha_2 \operatorname{Sat}_V(\mathbf{f}') \leq \alpha_2 \operatorname{Sat}_V(\mathbf{f}) \tag{7}$$

due to the assumption that the cardinalities of $\mathbf{f}$ and $\mathbf{f}'$ are equal. This gives us $\operatorname{Sat}_V(\mathbf{f}) \geq \operatorname{Sat}_V(\mathbf{f}')$ – a contradiction to the assumption that $\operatorname{Sat}_V(\mathbf{f}) < \operatorname{Sat}_V(\mathbf{f}')$.

Finally, it should be noted that as for the standard problem, it is often desirable that the owner fragment only consists of the attributes contained in singleton constraints and the respective tuple identifiers. To achieve this, one can again choose the weight of the owner fragment as explained in Lemma 1.

4.3 Dependencies

To model correlations between data, database dependencies can be specified. For example, in a medical setting a specific treatment might disclose the diagnosed disease. In a data publishing and also a data sharing application, such dependencies on the one hand enable *users* to infer more information from retrieved data; on the other hand, in a separation of duties setting, dependencies can enable a *server* to deduce much more information even though it only stores fragments of a confidentiality-preserving fragmentation: in the example, the specific disease can be inferred which however might be highly confidential information. Such inferences disclosing confidential information must be avoided and hence dependencies have to be considered when applying separation of duties.

De Capitani di Vimercati et al. [18] have explored the technique of fragmentation to ensure data confidentiality in presence of dependencies among columns. We will adopt their notion of dependencies that are specified as rules with a left-hand side (the premise) and a right-hand side (the consequence); both the premise and the consequence are sets of on column names. The intended semantics is that of a functional dependency: any combination of values for the premise uniquely discloses a combination of values for the consequence. Dependencies on the patient table could for example be $\mathsf{DoB}, \mathsf{ZIP} \rightsquigarrow \mathsf{Name}$ (that discloses the

name of a patient from the date of birth and zip code) or Treatment $\rightsquigarrow$ Diagnosis (that discloses a diagnosis from the treatment). De Capitani di Vimercati et al. explore this problem only in a single-relational environment. For our application, the definitions and theories will be translated into a multi-relational context.

Definition 8. ***(Data Dependency)*** *A* dependency δ *over a database schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ *is an expression of the form* $X \rightsquigarrow Y$*, with* $X, Y \subset \bigcup_{s=1}^{N} A_s$ *and* $X \cap Y = \emptyset$*. The left hand side of a dependency* δ *is called the premise while the right hand side is called the consequence of* δ*. For simplicity, the notations* δ*.premise and* δ*.consequence (or* δ*.p and* δ*.c, for short) are used to denote the respective part of the dependency.*

A simple approach to make the information disclosed by dependencies visible in a fragment is adding the implied attributes to the fragment (this approach is called fragment and dependency composition in [18]):

Definition 9. ***(Dependency Composition)*** *For a given database schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$*, a subset* $f_j \subseteq \bigcup_{s=1}^{n} A_s$ *of attributes and a set* Δ *of dependencies, the* composition *of* f_j *with dependency* $\delta \in \Delta$ *is the set of attributes:*

$$f_j \otimes \delta = \begin{cases} f_j \cup \delta.consequence, & \text{if } \delta.premise \subseteq f_j \\ f_j, & \text{else} \end{cases}$$

Next, we adopt the notion of closure of a set of attributes as in [18]; the closure is a superset that is immune to dependency composition:

Definition 10. ***(Closure)*** *Let* $A_1, \ldots, A_N$ *denote pairwise disjoint sets of attributes and let* $d = \{r_1, \ldots r_N\}$ *be a database state over the schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$*. Moreover, let* Δ *denote a set of dependencies. For any subset* $f \subseteq \bigcup_{s=1}^{n} A_s$ *of attributes the* closure *with respect to* δ *is defined as the minimal set* $\overline{f}$ *which satisfies* $f \subseteq \overline{f} \subseteq \bigcup_{s=1}^{N} A_s$ *and for all* $\delta \in \Delta$ *it holds that* $\overline{f} \otimes \delta = \overline{f}$*. If the subset* f *satisfies* $f = \overline{f}$ *it is called* closed.
If $\mathbf{f} = (f_0, \ldots, f_k)$ *denotes a lossless/correct fragmentation of* d*, the closure of that fragmentation with respect to* Δ *is defined as* $\overline{\mathbf{f}} \coloneqq (\overline{f}_0, \ldots, \overline{f}_k)$*. A fragmentation for which every server fragment* $f_j \in \{f_1, \ldots, f_k\}$ *is closed is called a closed fragmentation.*

It is generally not possible to find a *closed* correct fragmentation satifying the disjointness property. Hence, the following problem statement focuses on finding a closed lossless multi-relational fragmentation.

Definition 11. ***(Multi-relational Separation of Duties in Presence of Data Dependencies)*** *Given a database state* $d = \{r_1, \ldots, r_N\}$ *over the given database schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$*, tuple identifiers* $tid_s \subset A_s$ *for all* $s \in \{1, \ldots, N\}$*, a well-defined set of multi-relational confidentiality constraints* C*, visibility constraints* V*, a set of dependencies* Δ*, a weight function* w_d*, servers* $S_0, \ldots, S_k$ *with maximum capacities* $W_0, \ldots, W_k \in \mathbb{R}_0^+$ *and positive*

weights $\alpha_1, \alpha_2 \in \mathbb{R}_0^+$*, find a* ***closed*** *lossless confidentiality-preserving fragmentation* $\mathbf{f} = (f_0, \ldots, f_k)$ *of* d *which satisfies* $w_d(f_j) \leq W_j$ *for all* $0 \leq j \leq k$ *such that the weighted sum* $\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f})$ *is minimized.*

We now discuss the influence of dependencies on solvability of the Multi-relational Separation of Duties problem. One might wonder, whether the condition that the fragmentation is closed will prevent finding a solution when a non-closed confidentiality-preserving fragmentation exists that would solve the problem. As opposed to [18], in our problem statement this could be the case due to the capacity constraints. As every fragment f_j is a subset of its closure $\overline{f}_j$ it also holds that $w_d(f_j) \leq w_d(\overline{f}_j)$. For the Extended Multi-relational Separation of Duties Problem it was recommended to choose the capacity of the owner fragment such that it can only hold the attributes contained in singleton constraints and the respective tuple-identifier attributes (see Lemma 1). When dependencies are taken into account however, using this capacity could make the problem unsolvable because additional attributes should be stored in the owner fragment because they are sensitive on their own due to dependency. To illustrate this, a dependency $\delta \in \Delta$ with $|\delta.premise| = 1$ and $\delta.premise \neq c$ for all $c \in C$ is supposed. The premise contains a single attribute which is not contained in a singleton constraint. At first glance, it seems that this attribute is not sensitive on its own and therefore, the attribute in $\delta.premise$ will not be placed in the owner fragment when choosing the owner capacity W_0 as described in Lemma 1. Because the problem statement requires a closed fragmentation, the server fragment that contains the attribute in $\delta.premise$ needs to hold $\delta.consequence$, too. Obviously, a problem arises if there exists a confidentiality constraint $c \in C$ with $c \subseteq \delta.premise \cup \delta.consequence$ because the fragmentation obeying W_0 cannot be confidentiality-preserving and therefore, no solution exists. Such situations occur if the closure of an attribute is sensitive – in other words the attributes revealed by a single attribute due to dependencies violate a confidentiality constraint. Therefore, the actual set of sensitive attributes is given by the union of the two sets

$$A^* := \{c \in C \mid |c| = 1\}$$

and

$$A^{**} := \bigcup_{s=1}^{N} \left\{ a_i^s \in A_s \mid \exists c \in C \text{ with } c \subseteq \overline{\{a_i^s\}} \right\}.$$

There are two possible solutions for this. The first one is to introduce new confidentiality constraints $c = \{a_i^s\}$ for all $a_i^s \in A^{**}$ which is justified because those attributes can be regarded as sensitive attributes. The other solution is to increase the capacity of the owner fragment such that it holds all the attributes in A^* and A^{**} and the necessary tuple-identifier attributes to ensure the reconstruction property of the fragmentation. We chose the second solution and accomplished it by defining the capacity of the owner fragment as:

$$W_0 = \sum_{s:A_s \cap (A^* \cup A^{**}) \neq \emptyset} w_d(\mathrm{tid}_s) + \sum_{a \in (A^* \cup A^{**})} w_r(a),$$

With this setting, the owner fragment only stores the minimum amount of attributes necessary to ensure confidentiality.

5 ILP Formulation

In the following subsections, we discuss in detail how we translate the Multi-relational Separation of Duties Problem in Presence of Data Dependencies into an integer linear program (ILP) representation. All indicator variables z_v, u_{vj} (both for the visibility constraints), $p_{\delta j}$ (for the dependencies), y_j (for the fragments) and x_{ij}^s (for the attributes) are binary. Moreover, we establish the convention that $s \in \{1, \ldots, N\}$, $a_i^s \in A_s$ (the attributes), $a_{i'}^s \in \mathrm{tid}_s$ (the tuple identifiers), $j \in \{0, \ldots, k\}$, $c \in C$ (the confidentiality constraints), $v \in V$ (the visibility constraints), $\delta \in \Delta$ (the dependencies).

The overall ILP that results in a confidentiality-preserving closed fragmentation (according to confidentiality constraints $c \in C$ and dependencies $\delta \in \Delta$) that at the same time occupies a minimum amount of servers and maximizes the amount of satisfied visibility constraints $v \in V$ is shown in Figure 1.

$$\min \quad \alpha_1 \sum_{j=0}^{k} y_j - \alpha_2 \sum_{v \in V} z_v \tag{8}$$

$$\text{s.t.} \quad \sum_{j=0}^{k} x_{ij}^s \geq 1, \tag{9}$$

$$\sum_{a_i^s \in A_s^*} x_{ij}^s \leq x_{i'j}^s |A_s^*|, \tag{10}$$

$$\sum_{a_i^s \in A_s^*} x_{ij}^s \geq x_{i'j}^s, \tag{11}$$

$$\sum_{s=1}^{N} \sum_{a_i^s \in A_s} w_d(a_i^s) x_{ij}^s \leq W_j y_j, \tag{12}$$

$$\sum_{a_i^s \in c} x_{ij}^s \leq |c| - 1, \tag{13}$$

$$\sum_{a_i^s \in v} x_{ij}^s \geq u_{vj} |v|, \tag{14}$$

$$\sum_{j=0}^{k} u_{vj} \geq z_v, \tag{15}$$

$$\sum_{a_i^s \in \delta.p} x_{ij}^s \leq |\delta.p| - 1 + p_{\delta j}, \tag{16}$$

$$\sum_{a_i^s \in \delta.p} x_{ij}^s \geq p_{\delta j} |\delta.p|, \tag{17}$$

$$\sum_{a_i^s \in \delta.c} x_{ij}^s \geq p_{\delta j} |\delta.c|, \tag{18}$$

Fig. 1. Integer Linear Program

A solution to the Separation of Duties Problem in Presence of Data Dependencies can be derived from a solution to the ILP by constructing the fragments according to the following rule. Attributes a_i^s for which the corresponding variable x_{ij}^s is equal to 1 in the ILP solution are contained in fragment j provided that the fragment j shall be non-empty (which is denoted by $y_j = 1$ in the ILP

solution).

$$f_j \coloneqq \begin{cases} \bigcup_{s=1}^{N} \{a_i^s \in A_s \mid x_{ij}^s = 1\}, & \text{if } y_j = 1 \\ \emptyset, & \text{else} \end{cases}$$

5.1 Translating confidentiality

Indicator variables $x_{ij}^s \in \{0,1\}$ are used to express that attribute $a_i^s \in A_s$ from relation r_s is placed on server j. For every $s \in \{1,\ldots,k\}$ let $A_s^* \coloneqq A_s \setminus \text{tid}$ denote the set of non-tuple identifiers of A_s. Binary variables $y_1,\ldots,y_k \in \{0,1\}$ are introduced which take a value of one if fragment f_j should be non-empty and a value of zero otherwise. In the objective function (8) the expression $\alpha_1 \sum_{j=0}^{k} y_j$ minimizes the cardinality of the fragmentation. Condition (9) ensures that every attribute is placed in at least one fragment satisfying the completeness property. Constraint (10) conditions that if there is a non-tuple identifier attribute contained in fragment f_j, i.e. the left hand side of the inequality is greater than one, then the right hand side must equal $|A_s^*|$ which is fulfilled if the variable $x_{i'j}^s$ for the tuple identifier attribute $a_{i'}^s \in \text{tid}_s$ also equals one and is hence also part of the fragment. The definition of the reconstruction property of multi-relational fragmentation requires that the tuple identifiers tid_s are proper subsets of the fragments; Constraint (11) takes care of this by allowing the variables $x_{i'j}^s$ for attribute $a_{i'}^s \in \text{tid}_s$ to equal one if at least one variable x_{ij}^s belonging to a non-tuple identifier attribute $a_i^s \in A_s^*$ equals one. Constraint (12) makes sure that the storage capacities are not exceeded and that y_j must take a value of one, whenever any attribute a_i^s is included in fragment f_j. Lastly Condition (13) is used to guarantee a confidentiality-preserving fragmentation, because at most $c-1$ variables x_{ij}^s for $a_i^s \in c$ can be equal to one.

5.2 Translating visibility

Additional binary variables u_{vj} are introduced for every *visibility* constraint $v \in V$ and fragment $j \in \{1,\ldots,k\}$. These variables should only take a value of one if all attributes contained in v are placed in fragment f_j. Moreover, indicator variables z_v for all visibility constraints $v \in V$ are used to indicate whether there is at least one fragment that contains all attributes of visibility constraint $v \in V$. In the objective function (8) the summand $-\alpha_2 \sum_{v \in V} z_v$ maximizes the number of satisfied visibility constraints.

The constraints that ensure the proper treatment of the visibility constraints are given by Conditions (14) and (15). The former ensures that for every fragment f_j and every visibility constraint $v \in V$ variable u_{vj} can only be equal to one if the visibility constraint is satisfied in fragment f_j. The latter then makes sure that z_v, the indicator variable for visibility constraint $v \in V$, can only be equal to one if there is at least one u_{vj} for $j \in \{1,\ldots,k\}$ that equals one, i.e. visibility constraint v is satisfied one at least on server.

5.3 Translating dependencies

It can easily be seen that a server fragment f_j equals its closure $\overline{f}_j$ if and only if for every data dependency $\delta \in \Delta$ one of the conditions $\delta.p \not\subseteq f_j$ or $\delta.p \cup \delta.c \subseteq f_j$ is true [18]. Hence, to check whether a server fragment f_j is closed, one first has to check for every data dependency $\delta \in \Delta$ if $\delta.p \in f_i$. In the ILP formulation we introduce indicator variables $p_{\delta j} \in \{0, 1\}$ for each data dependency $\delta \in \Delta$ and server $j \in \{1, \ldots, k\}$ that should take a value of one if and only if all attributes in $\delta.p$ are stored in server fragment f_j. After that, we have to make sure that if a dependency premise is contained in a fragment the consequence must also be contained. Together, Constraints (16) and (17) ensure that $p_{\delta j}$ equals one if and only if all attributes in $\delta.p$ should be placed into the same server fragment f_j. The sum on the left hand side of Condition (16) is at most $|\delta.p|$. If this is the case, then $p_{\delta j}$ must equal one because otherwise the expression on the right hand side would be smaller. Hence, if all attributes in the premise of δ are part of fragment f_j variable $p_{\delta j}$ must take a value of one. Furthermore, Condition (17) achieves that variable $p_{\delta j}$ will be zero otherwise. If the left hand side of the equality is smaller than $|\delta.p|$, i.e. not all attributes in the premise of δ are contained in fragment f_j, then constraint can only be fulfilled if the right hand side equals zero or in other words $p_{\delta j}$ equals zero.

Finally, Constraint (18) requires that all attributes in $\delta.c$ are part of server fragment f_j if all attributes in $\delta.p$ are part of f_j: if $p_{\delta j}$ is equal to one the right hand side of the inequality takes the value $|\delta.c|$. In this case the constraint can only be fulfilled if the sum on the right side is also $|\delta.c|$ which means that x^s_{ij} equals one for all $a^s_i \in \delta$. On the other hand if $p_{\delta j}$ equals zero the condition is always fulfilled.

6 Implementation

There are the following entities involved in the system:

- **Untrusted Database Servers:** These servers store the server fragments and can process queries involving their respective fragment only. The individual physical fragments are organized in database tables.
- **Trusted Database Server:** This server stores the owner fragment and manages connections to the untrusted servers. Most common DBMSs provide means to include database tables stored at remote servers. In PostgreSQL for example, this can be realized with so-called *Foreign Data Wrappers* and MySQL provides the *FEDERATED Storage Engine*. This enables the later presented distributed database client to issue adequate high-level (possibly SQL) queries directly to the trusted database server instead of issuing subqueries to each individual database server and then calculating the desired result. Instead, the built-in query processor of the trusted database server will decide how the query is actually optimized and executed.
- **Distributed Database Client:** The client acts as an additional layer between the database users and the database servers. It computes the fragmentation using an ILP solver, stores the metadata of the fragmentation (i.e. at

which server the columns are stored) and processes and rewrites user queries such that they are based on fragments instead of relations of the original database. The distributed database client can either access the database servers directly if a query exclusively involves columns of a single fragment or it can issue queries to the trusted database server which makes all other fragments stored at the untrusted databases servers visible. Those queries are then analyzed by the database system's query processor which decides how the query is internally executed by applying adequate optimization techniques. Finally, the results of the queries are transferred to the user.

The major advantage of the presented framework is that the only component that we had to implement is the distributed database client (available at [5]) that relies on the advanced query optimization techniques already provided by today's DBMS. The chosen programming language is JAVA and the implementation relies on the popular open source DBMS PostgreSQL. Therefore, the trusted database server and the untrusted database servers need to run a standard installation of PostgreSQL. To solve the ILPs the IBM ILOG CPLEX solver is used due to its comprehensive range of available solving strategies and several off-the-shelf optimizations. Lastly, to analyze and rewrite the users' queries, the distributed database client uses the open source project JSQLParser which is a SQL parser for JAVA. After solving the ILP using the CPLEX solver, the distributed database client continues with creating databases on the necessary database servers to store the respective physical fragments. In particular, a new database is set up at the trusted database server to store the owner fragment. Subsequently, the table fragments are set up and populated with the data from the original database. Using the foreign data-wrapper module postgres_fdw, the tables stored at the untrusted servers are made visible in the newly created database at the trusted server. Therefore, the database at the trusted database server contains each of the table fragments either as a local table if the fragment is part of the owner's physical fragment or as foreign table, otherwise. As a final step, the distributed database client sets up views in the database of the trusted database server using the local and foreign tables which correspond to the tables of the original database.

We now describe in detail the functionality that is offered by our system. To illustrate the individual steps, a small database consisting of two tables in a hospital scenario serves as a running example. The first table stores information about patients and the second table stores information about doctors working in the hospital:

Setup. For the setup, the owner has to set up the database to be fragmented at the trusted database server and specify the designated tuple identifier columns of the table as primary key columns. Furthermore, the owner has to provide a configuration file to tell the distributed database client where the database servers are located and how much space is available on each server; see Figure 2 for an example with one owner server and three remote servers. Additional files can be created to specify the confidentiality constraints, visibility constraints and data dependencies (see Figures 3, 4 and 5). In contrast to the configuration

```
Name, Address,        Port, Username, Password, Capacity, IsOwner
S0,   192.168.178.92, 5432, postgres, postgres, 2.0,      owner;
S1,   192.168.178.82, 5432, postgres, postgres, INF;
S2,   192.168.178.87, 5432, postgres, postgres, INF;
S3,   192.168.178.88, 5432, postgres, postgres, INF;
```

Fig. 2. Example configuration file

```
patient.P_Name;
doctor.D_Name;
patient.P_DoB, patient.P_ZIP, patient.P_Diagnosis;
doctor.D_DoB, doctor.D_ZIP;
```

Fig. 3. Example confidentiality constraints

```
patient.P_Diagnosis, patient.P_ZIP;
patient.P_Treatment, doctor.D_Specialty;
```

Fig. 4. Example visibility constraints

```
patient.P_DoB, patient.P_ZIP > patient.P_Name;
patient.P_Treatment > patient.P_Diagnosis;
patient.P_Diagnosis > patient.P_Treatment;
doctor.D_DoB, doctor.D_ZIP > doctor.D_Name;
```

Fig. 5. Example dependencies

P_PatientID	P_Name	P_DoB	P_ZIP	P_Diagnosis	P_Treatment	P_DoctorID
1	J. Doe	07.01.1986	12345	Flu	Nose spray	1
2	W. Lee	12.08.1974	23456	Broken Leg	Gypsum	2
3	F. Jones	05.09.1963	23456	Asthma	Asthma inhaler	1
4	G. Miller	10.02.1982	12345	Cough	Cough syrup	1

Table 1. Table *patient* with primary key column *P_PatientID*

D_DoctorID	D_Name	D_DoB	D_ZIP	D_ Specialty
1	H. Bloggs	04.02.1971	34567	Respiratory
2	G.Douglas	27.07.1965	23456	Fraction

Table 2. Table *doctor* with primary key column *D_DoctorID*

file, those files are optional. Subsequently, the owner instructs the distributed database client to set up the vertically fragmented database by solving an optimization problem with the CPLEX solver based on the provided input data. The weights of the specific columns are computed automatically and do not have to be provided by the owner. After solving the ILP using the CPLEX solver, the distributed database client continues with creating databases on the necessary database servers to store the respective fragments. In particular, a new database is set up at the trusted database server to store the owner fragment. Subsequently, the tables are populated with the data from the original database. In the example, the table fragments shown in Table 3 are stored in the owner fragment at server S_0. Server S_1 stores the table fragments shown in Table 4. Lastly, Server S_2 stores the table fragments shown in Table 5. We implemented this functionality using the foreign data-wrapper module postgres_fdw which makes the server fragments available via the trusted database server as foreign tables. As a final step, the distributed database client sets up views in the database of the trusted database server using the local and foreign tables which correspond to the tables of the original database as shown in Figure 6.

SELECT. The distributed database client provides two possible ways of querying the vertically fragmented database. The first possibility involves explicitly **rewriting** the users' queries using the JSQLParser such that they act on table fragments instead of the original tables. The second possibility is based on the created **views**.

The advantage of explicitly rewriting the users' queries is that the distributed database client can analyze the queries to make educated decisions which columns and table fragments are actually involved in the query and omit those that are not. In the provided implementation, for each table involved in the query, the distributed database client assesses which columns are needed and greedily chooses table fragments to obtain all necessary columns. As an example of a rewritten query, the SQL statement in Figure 7 is rewritten inside the distributed database client into the SQL query shown in Figure 8.

There are two major points to notice in this query. First, only servers S_0 and S_1 take part in the query, i.e. the fragments stored on S_3 are not affected.

P_PatientID	P_Name
1	J. Doe
2	W. Lee
3	F. Jones
4	G. Miller

D_DoctorID	D_Name
1	H. Bloggs
2	G. Douglas

Table 3. Owner Fragment

P_PatientID	P_DoB	P_DoctorID
1	07.01.1986	1
2	12.08.1974	2
3	05.09.1963	1
4	10.02.1982	1

D_DoctorID	D_ZIP
1	34567
2	23456

Table 4. Server fragment 1

Furthermore, the WHERE condition is pushed down in to the SELECT queries affecting the table fragments as far as it can be processed by the respective server. This could potentially lead to less data being transferred during the execution process when tuples that do not satisfy the condition are excluded. This example also shows the major drawback of explicitly rewriting the queries which lies in the complexity of SQL that makes query rewriting a very challenging task.
In contrast, using views over the table fragments to recreate the original tables is a simple strategy to avoid query rewriting. The queries remain unchanged but logically they involve the views instead of real physical tables. Therefore, it is up to PostgreSQL's query processor to decide how these queries are executed. While this method is easy to implement, its major drawback lies in the fact that each query involves all the columns of each table and unnecessary table fragments are not omitted from the query. For example, a query that selects only one column of a table fragmented among three servers will always involve all three of the servers although one would be sufficient. To the best of our knowledge, the PostgreSQL query processor does currently not consider excluding JOIN clauses of tables that are not affected by the query.

INSERT. When the distributed database client receives a request to insert a specific row into a table, it determines the affected table fragments and inserts a row, restricted to the according columns, in each of those. In the example, an

P_PatientID	P_ZIP	P_Diagnosis	P_Treatment
1	12345	Flu	Nose spray
2	23456	Broken Leg	Gypsum
3	23456	Asthma	Asthma inhaler
4	12345	Cough	Cough syrup

D_DoctorID	D_DoB	D_ Specialty
1	04.02.1971	Respiratory
2	27.07.1965	Fraction

Table 5. Server fragment 2

```
CREATE OR REPLACE VIEW doctor
AS
  SELECT s2_doctor_frag.d_doctorid  AS d_doctorid,
         s0_doctor_frag.d_name      AS d_name,
         s2_doctor_frag.d_dob       AS d_dob,
         s1_doctor_frag.d_zip       AS d_zip,
         s2_doctor_frag.d_specialty AS d_specialty
  FROM   s2_doctor_frag
         LEFT JOIN s0_doctor_frag USING(d_doctorid)
         LEFT JOIN s1_doctor_frag USING(d_doctorid);

CREATE OR replace VIEW patient
AS
  SELECT s2_patient_frag.p_patientid AS p_patientid,
         s0_patient_frag.p_name      AS p_name,
         s1_patient_frag.p_dob       AS p_dob,
         s2_patient_frag.p_zip       AS p_zip,
         s2_patient_frag.p_diagnosis AS p_diagnosis,
         s2_patient_frag.p_treatment AS p_treatment,
         s1_patient_frag.p_doctorid  AS p_doctorid
  FROM   s2_patient_frag
         LEFT JOIN s0_patient_frag USING(p_patientid)
         LEFT JOIN s1_patient_frag USING(p_patientid);
```

Fig. 6. View creation

```
SELECT  p_name, p_diagnosis, d_name AS attending_doctor
FROM patient, doctor
WHERE patient.p_doctorid = doctor.d_doctorid AND p_name = 'J. Doe';
```

Fig. 7. Original SELECT statement

INSERT statement of the form shown in Figure 9 is translated into the three INSERT statements shown in Figure 10.

DELETE. Deletion of rows can be broken down into two parts. Because table fragments only contain a subset of columns, it is generally not possible to evaluate the condition specified by the WHERE clause of the SQL DELETE statement with a single table fragment. Therefore, as the first part, a SELECT query is used to detect the values of the tuple identifiers of the rows that should be deleted. This SELECT query is executed by the distributed database client by either using the defined views or explicitly creating an according SELECT statement based on table fragments. The result of this query is stored in a temporary table. Subsequently, DELETE queries are executed for each table fragment with the condition that the tuple identifier values are present in the temporary table. Finally, the temporary table is deleted. In the prototype implementation, this is done by putting all of those queries into a single transaction, which is a sequence

```
SELECT p_name, p_diagnosis, d_name AS attending_doctor
 FROM (SELECT
 patientS0.p_name      AS p_name,
 patientS0.p_patientid AS p_patientid,
 patientS2.p_diagnosis AS p_diagnosis,
 patientS1.p_doctorid  AS p_doctorid
  FROM  (SELECT p_name, p_patientid
   FROM   s0_patient_frag
   WHERE  ( true AND p_name = 'J. Doe' )) AS patientS0
   INNER JOIN (SELECT p_diagnosis, p_patientid
    FROM   s2_patient_frag
    WHERE  ( true AND true )) AS patientS2
    ON patientS0.p_patientid =
    patientS2.p_patientid
   INNER JOIN (SELECT p_doctorid, p_patientid
    FROM   s1_patient_frag
    WHERE  ( true AND true )) AS patientS1
    ON patientS2.p_patientid = patientS1.p_patientid) AS patient,
    (SELECT
    doctorS0.d_doctorid AS d_doctorid,
    doctorS0.d_name     AS d_name
    FROM  (SELECT d_doctorid, d_name
     FROM   s0_doctor_frag
     WHERE  ( true AND true )) AS doctorS0) AS doctor
     WHERE  patient.p_doctorid = doctor.d_doctorid AND p_name = 'J. Doe';
```

Fig. 8. Rewritten SELECT statement

```
INSERT INTO doctor
VALUES (3, 'C. Hall', '12/11/1990', 12345, 'Dermatology');
```

Fig. 9. Original INSERT statement

of SQL statements that is being executed consecutively and if specified, automatically deletes the created temporary tables at the end of the transaction. To illustrate this, the DELETE statement in Figure 11 results in the transaction shown in Figure 12. Note that the keyword TEMP specifies a temporal table and the ON COMMIT DROP option specifies that the temporal table is deleted at the end of the transaction.

UPDATE. Performing updates on rows of a specific vertically fragmented table resembles the deletion of rows due to the fact that the WHERE condition has to be specified. Therefore, as the first step, a temporal table is created that stores the tuple identifier columns of the affected rows. Subsequently, the distributed database client determines the involved table fragments and performs an update operation on each of those. To make sure that the proper rows are updated, the condition that the tuple identifier values are present in the temporal

```
INSERT INTO s0_doctor_frag (d_doctorid,d_name)
VALUES (3, 'C. Hall');

INSERT INTO s2_doctor_frag (d_doctorid, d_dob, d_specialty)
VALUES (3, '12/11/1990', 'Dermatology');

INSERT INTO s1_doctor_frag (d_doctorid,d_zip)
VALUES (3, 12345);
```

Fig. 10. Rewritten INSERT statement

```
DELETE FROM doctor WHERE d_name='G. Douglas';
```

Fig. 11. Original DELETE statement

table is enforced. Those operations are again executed in a single transaction and the temporal table is dropped when the transaction is committed. Consider the UPDATE query in Figure 13 for which the resulting transaction is shown in Figure 14.

7 Evaluation

The prototype implementation is tested with two popular TPC benchmarks (TPC-E and TPC-H) for databases [29, 30]. Each of these benchmarks is suitable to evaluate different aspects of the Separation of Duties Problem. The database defined by the TPC-E benchmark consists of 33 tables and a total number of 191 columns; we only use the TPC-E schema which due to its size it is suitable to evaluate the effects of the number of constraints and dependencies on the performance of the ILP solver and the resulting fragmentations. In contrast, from the TPC-H benchmark we use data as well as queries in order to test the distributed runtime performance of our approach. It consists of a database containing 8 tables that models a typical database in a business environment. Moreover, it defines 22 complex SQL queries that are typical in decision support scenarios. Therefore, this benchmark is well-suited to test the implementation's capabilities in terms of query processing.

All of the tests were executed on a single PC equipped with an Intel Xeon E3-1231v3 @ 3.40GHz (4 Cores), 32GB DDR3 RAM and a Seagate ST2000DM001 2TB HDD with 7200 rpm. The PC is running Ubuntu 16.04 LTS. The database servers – including the trusted database server hosting the owner fragment – are running in separate, identical virtual machines which are assigned 4 cores and 8GB of RAM. The virtual machines are running Ubuntu Server 16.04 LTS with an instance of PostgreSQL 9.6.1 installed. By running the servers in identical virtual machines, it is guaranteed that the results are not influenced by hardware or software differences. Lastly, the CPLEX version used by the implementation is CPLEX 12.7.

```
START TRANSACTION;
CREATE TEMP TABLE tmpdelete ON COMMIT DROP AS
 (SELECT d_doctorid
  FROM (SELECT doctors0.d_doctorid AS d_doctorid,
               doctors0.d_name     AS d_name
   FROM (SELECT d_doctorid, d_name
    FROM s0_doctor_frag
     WHERE  d_name = 'G. Douglas') AS doctors0) AS doctor
   WHERE  d_name = 'G. Douglas');

DELETE FROM s0_doctor_frag
WHERE (d_doctorid) IN (SELECT * FROM tmpdelete);

DELETE FROM s2_doctor_frag
WHERE (d_doctorid) IN (SELECT * FROM tmpdelete);

DELETE FROM s1_doctor_frag
WHERE (d_doctorid) IN (SELECT * FROM tmpdelete);

COMMIT;
```

Fig. 12. Rewritten DELETE statement

```
UPDATE patient SET p_zip = 23456 WHERE p_name = 'G. Miller';
```

Fig. 13. Original UPDATE statement

7.1 TPC-E Data Set

The TPC-E benchmark is intended to model the workload of a brokerage firm. It consists of 33 database tables which fall into four categories [29]:

- **Customer tables**: There are 9 tables that contain information about the brokerage firm's customers.
- **Broker tables**: There are 9 tables that contain information about the brokerage firm.
- **Market tables**: There are 11 tables that contain information about companies, markets, exchanges and industry sectors.
- **Dimension tables**: There are 4 tables that contain common information like zip codes or addresses.

Using the TPC-E database schema, our tests explore the influence of different sizes of sets of well-defined confidentiality constraints, visibility constraints and dependencies on the solver's performance and the resulting fragmentation. The trusted database server's capacity is set to zero: it should not store any data; this means that for the tests, singleton constraints are disallowed in the confidentiality constraints; moreover, the case that an association constraint contains an attribute that (after applying all possible dependencies) has an entire

```
START TRANSACTION;
CREATE TEMP TABLE tmpupdate ON COMMIT DROP AS
 (SELECT p_patientid FROM
  (SELECT patientS0.p_patientid AS p_patientid,
          patients0.p_name      AS p_name
    FROM (SELECT p_patientid, p_name
     FROM s0_patient_frag WHERE p_name = 'G. Miller')
          AS patientS0) AS patient
   WHERE p_name = 'G. Miller');

UPDATE s2_patient_frag SET p_zi=23456
 WHERE (p_patientid)IN (SELECT * FROM tmpupdate);
COMMIT;
```

Fig. 14. Rewritten UPDATE statement

confidentiality constraint in its closure is disallowed, too: more formally, we disallow attributes a such that there is a constraint c with $c \subseteq \bar{a}$. This restriction is introduced to allow a maximal number of possible choices for the placement of the attributes during the optimization process because attributes contained in singleton constraints or single attributes with a sensitive closure can only be placed in the owner fragment which would limit the number of decisions the ILP solver has to draw.

Settings. The constraints and dependencies are generated randomly. To measure scalability of the approach, we introduce scale factors $\sigma_C, \sigma_V, \sigma_\Delta \in \mathbb{R}_0^+$ for confidentiality constraints, visibility constraints and dependencies, respectively. Note that the overall number of non-primary key columns is $n = 142$ in the TPC-E database. The scale factors can be interpreted as constraints/dependencies per non-primary key column. Because the primary-key columns act as tuple identifiers, they are supposed to be insensitive and are therefore neither part of the constraints nor part of the premise or consequence of dependencies.

Hence, if $A^* = \bigcup_{s=1}^{33} A_s \setminus \mathrm{tid}_s$ denotes the set of all non-primary key columns of the 33 tables and $\mathcal{P}(\mathcal{A}^*)$ denotes its powerset, the selection process is carried out as follows:

- **Confidentiality Constraints**: For each of the designated cardinalities $\nu \in \{2, 3, 4, 5\}$, sets of confidentiality constraints $C_\nu \subseteq \mathcal{P}(\mathcal{A}^*)$ of cardinality $|C_\nu| = \lceil \frac{n}{4} \cdot \sigma_C \rceil$ are selected randomly from A^*. For each ν and C_ν it holds that $|c| = \nu$ for all $c \in C_\nu$. The resulting set of confidentiality constraints is then given by $C \coloneqq \bigcup_{\nu=2}^{5} C_\nu$. Moreover, during the generation of the confidentiality constraints it is ensured that the resulting set C is well-defined. Therefore, this process results in a well-defined set of confidentiality constraints C of cardinality $|C| = 4 \cdot \lceil \frac{n}{4} \cdot \sigma_C \rceil$ which is equally divided into confidentiality constraints of cardinality 2, 3, 4 and 5. A scale factor of $\sigma_C = 1$ would therefore result in a set of 144 confidentiality constraints

which corresponds roughly to the size of non-primary key attributes in the database. As confidentiality constraints with lower cardinality are generally harder to satisfy than constraints with high cardinality, restricting ν to the values $\{2, 3, 4, 5\}$ is not a serious limitation.

- **Visibility Constraints**: Generating the visibility constraints is carried out similarly to the generation of confidentiality constraints. The only difference in the process is that the resulting set does not have a limitation of being well-defined. Hence, for each of the cardinalities $\nu \in \{2, 3, 4, 5\}$, sets of visibility constraints $V_\nu \subseteq \mathcal{P}(\mathcal{A}^*)$ of cardinality $\lceil \frac{n}{4} \cdot \sigma_V \rceil$ are selected randomly such that $|v| = \nu$ for all $v \in V_\nu$. The overall set of visibility constraints is hence given by $V \coloneqq \bigcup_{\nu=2}^{5} V_\nu$ which has a cardinality of $|V| = 4 \cdot \lceil \frac{n}{4} \cdot \sigma_V \rceil$ and is equally divided into visibility constraints of cardinality 2, 3, 4 and 5.
- **Dependency**: Sampling dependencies is carried out differently because a dependency $\delta = \delta.premise \rightsquigarrow \delta.consequence$ is defined by the two sets $\delta.premise$ and $\delta.consequence$. The scale factor σ_Δ determines the cardinality of the set of dependencies Δ which is given by $|\Delta| = \lceil n \cdot \sigma_\Delta \rceil$. The dependencies itself are generated iteratively as follows: First, two random values $\nu_p \in \{2, \dots 5\}$ and $\nu_v \in \{1, \dots, 5\}$ are determined which define the cardinalities of the premise and the consequence. Then, $\delta.premise \subseteq \mathcal{P}(\mathcal{A}^*)$ and $\delta.consequence \subseteq \mathcal{P}(\mathcal{A}^*)$ are chosen randomly such that $|\delta.premise| = \nu_p$ and $|\delta.consequence| = \nu_c$. This process is executed $\lceil n \cdot \sigma_\Delta \rceil$ times to obtain the final set of dependencies. The cardinality of the premise of each dependency is restricted to a value between 2 and 5 because on the one hand a premise of cardinality one could make a single attribute sensitive and on the other hand if the cardinality gets too large, it will become easier to place the attributes into multiple fragments such that the dependency only slightly influences the resulting fragmentation. Moreover, the cardinality of the consequence of each dependency is restricted to be smaller than 5 to guarantee a moderate balance between the cardinality of the premise of a dependency and the impact it has on the resulting fragmentation in terms of its consequence.

For each of the executed test runs the weights α_1 and α_2 required by the problem statement of the Multi-relational separation of duties problem in presence of data dependencies are chosen such that they satisfy the inequality $\alpha_2|V| < \alpha_1$ presented in Lemma 2. Therefore, the resulting fragmentation shall be of minimal cardinality and the number of satisfied visibility constraints shall be maximal among all feasible fragmentations of minimal cardinality.

We executed several test runs with these settings. For all test runs a time limit of 30 minutes is set for the optimization process. Previous tests have shown that after this time a feasible solution can be found but the objective value does not significantly improve after this time.

Furthermore, different measurements are introduced to measure the quality of the resulting fragmentation. These measurements are based on the *objective value* of the best integer solution obj_I found by CPLEX and the *lower bound* obj_{LP} on the objective value which could be established by CPLEX during the

optimization progress by solving the LP-Relaxation of different subproblems of the ILP; the LP-Relaxation of an ILP is obtained by allowing the variables to take continuous instead of integral values. These measurements are defined as follows:

- **Relative MIP gap**: The *relative MIP gap* is a well-known general expression used by ILP solvers such as CPLEX to measure the quality of calculated solutions of mixed integer linear programs, i.e. linear programs of which some of the variables are restricted to be integer and others are real valued. Of course, it is also applicable for the special class of integer linear programs and it is defined by the following expression:

$$\frac{|obj_I - obj_{LP}|}{|obj_I|}$$

  The *relative MIP gap* measures the percentage of how much the objective value of an optimal solution can maximally deviate from the objective value obj_I of a feasible solution due to the established lower bound. Therefore, if this measure equals p, there is an uncertainty whether the objective value could potentially be reduced by up to p percent. Our overall objective function considers both the minimization of the cardinality (number of external servers) as well as the satisfaction of the visibility constraints.
- **Card gap**: We introduce this measure specifically for the Separation of Duties Problem to account for the quality of a feasible solution's fragmentation and it is defined as follows:

$$\left\lfloor \frac{|obj_I - obj_{LP}|}{|\alpha_1|} \right\rfloor$$

  The purpose of this expression is to measure the uncertainty about the fragmentation's cardinality of a feasible solution. For example, if this expression equals one, then it might be possible to reduce the fragmentation's cardinality by one and potentially, one database server less is necessary. If this expression equals zero, the fragmentation's cardinality is minimal and the number of servers necessary cannot be reduced.
- **Sat gap**: Similarly, we introduce this measure to account for the uncertainty of a solution in terms of visibility constraints with the following expression:

$$\left\lfloor \frac{|obj_I - obj_{LP}|}{|\alpha_2|} \right\rfloor$$

  If this expression equals zero, the number of satisfied visibility constraints is maximal for the feasible solution. Further, if this expression equals $n \geq 1$, it is uncertain, whether up to n more visibility constraints could potentially be satisfied.

If all of these measures are equal to zero for a feasible solution's objective value obj_I and the established lower bound obj_{LP}, an *optimal* solution has been found. However, these expressions require a good feasible solution on the one hand, and a good lower bound on the other hand and if either of those cannot be found a high uncertainty remains.

Test runs To study the impact of confidentiality constraint, visibility constraints and dependencies individually, the evaluation is structured into several test cases.

First, the effects of increasing the number of confidentiality constraints is studied; hence, the scale factors σ_V and σ_Δ are set to zero (no visibility constraints and no dependencies) and different values for σ_C are tested (I). Next, σ_C is set to four, σ_Δ is set to zero and different scale factors σ_V are used to test the influence of an increasing number of visibility constraints (II). Lastly, to test the effects of the number of dependencies σ_C is set to four, σ_V is set to 0.25 and different scale factors σ_Δ are evaluated (III).

For the first test case (I), the σ_V and σ_Δ is set to zero and the number of confidentiality constraints is increased with the scale factor $\sigma_C \in \{1, 2, 4, 8, 16\}$. The results of these test runs are summarized by Table 6. An optimal solution is

σ_C	$\lvert C\rvert$ (# conf. constraints)	Cardinality (# servers)	MIP gap	Card gap	Optimal?	Time (s)
1	144	2	0%	0	yes	0.61
2	284	3	0%	0	yes	1.79
4	568	3	0%	0	yes	1.55
8	1136	4	0%	0	yes	11.99
16	2272	5	16%	0	in 1 of 5 runs	in 1 run 297.27 (otherwise timeout)

Table 6. Increasing number of confidentiality constraints (average over 5 runs)

found nearly all of the scenarios as the *relative MIP gap* and the *Card gap* shows; that is, it is not possible to find a solution with less external servers. As the scale factor increases, more confidentiality constraints can only be satisfied when the individual original tables are split into more fragments (that is, the cardinality of the fragmentation increases). In the scenario with 2272 confidentiality constraints in four out of five cases we met our timeout limit of 1800 seconds and stopped the execution of CPLEX. Each of the fragments of one original table has to be stored on a server separate from the other servers storing fragments of the same original table; thus, the number of necessary database servers increases, too: for a scale factor of $\sigma_C = 1$ (that is, 144 confidentiality constraints), two servers are sufficient, for scale factors $\sigma_C = 2$ (284 confidentiality constraints) and $\sigma_C = 4$ (568 confidentiality constraints), three servers are necessary, for $\sigma_C = 8$ (1136 confidentiality constraints), there have to be four servers and for $\sigma_C = 16$ (2272 confidentiality constraints), five database servers have to be used. The runtime to find the optimal solution increases significantly for a scale factor of $\sigma_C = 16$; further optimizations of the solver could be employed to speed this setting up.

For the second test case (II), only the scale factor σ_V is changed and $\sigma_C = 4$ (568 confidentiality constraints) and $\sigma_\Delta = 0$ (no dependencies) remain fixed. In

other words, the effects of increasing the number of visibility constraints are evaluated. For all of the runs, the resulting fragmentation has a cardinality of three (which is minimal); because the weights α_1 and α_2 have been chosen according to Lemma 2, the visibility constraints do not affect the cardinality of the fragmentation. The overall results of test runs are presented in Table 7. When the

σ_V	$\lvert V\rvert$(#*vis.* constraints)	Sat	Cardinality (# servers)	MIP gap	Sat gap	Optimal?	Time (s)
0.25	36	27/36	3	0%	0	yes	101.84
0.5	72	44/72	3	1.83%	7.4	no	timeout
1	144	68/144	3	4.08%	33.6	no	timeout
2	284	105/284	3	6.85%	113.6	no	timeout

Table 7. Increasing number of visibility constraints (average over 5 runs)

number of introduced visibility constraints increases, the percentage of satisfied constraints decreases (see column "Sat"). Note that not all visibility constraints can be satisfied because they are conflicting with confidentiality constraints. The column "Sat gap" tells us how many more visibility constraints could potentially be satisfied in an optimal solution.

The most important thing to notice is that only the scenario with the lowest number of visibility constraints ($\sigma_V = 0.25$ corresponds to 36 visibility constraints) can be solved optimally and for this scenario the time increases significantly compared to the same scenario without visibility constraints (see $\sigma_C = 4$ in Table 6). The other three scenarios (72, 144, 284 visibility constraints, respectively) exceeded the time limit and were canceled without having found an optimal solution in terms of number of fragments and satisfied visibility constraints. One way to improve the results could therefore be to develop provably good heuristics to provide good starting solutions for the solver on the one hand and on the other hand, to establish tight lower bounds to point the solver in the right direction and allow less choices for the variables. Moreover, what is also an important conclusion of these results is that visibility constraint should not be viewed as a means to allow the execution of as much queries as possible on a single server. Rather, they should be used selectively to speed up a small amount of queries that are particularly relevant for the database.

Finally for the third test case (III), the effects of increasing the number of data dependencies are analyzed (see Table 8). For that, the scale factors $\sigma_C = 4$ (568 confidentiality constraints) and $\sigma_V = 0.25$ (36 visibility constraints) are fixed and the scale factors $\sigma_\Delta \in \{1, 2, 4, 8, 16\}$ (corresponding to 142, 284, 568, 1136 and 2272 dependencies, respectively) are used for the data dependencies. Hence, these results resemble very much the scenario with $\sigma_C = 4$ and $\sigma_V = 0.25$ of the previous test runs with the additional introduction of data dependencies: All of the the scenarios are solved optimally. A noticeable result is that increasing the number of data dependencies can in fact reduce the time needed to solve

σ_Δ	$\lvert\Delta\rvert$ (# dependencies)	Card (# servers)	Sat	MIP gap	Optimal?	Time (s)
1	142	3	26.4/36	0%	yes	60.22
2	284	3	27.2/36	0%	yes	75.91
4	568	3	27/36	0%	yes	41.36
8	1136	3	25/36	0%	yes	47.18
16	2272	3	23.6/36	0%	yes	21.57

Table 8. Increasing number of data dependencies (average over 5 runs)

the problem. An important take-away message from these test runs is that the solver benefits from introducing data dependencies instead of using excessively many confidentiality constraints.

7.2 TPC-H Data Set

The TPC-H benchmark is described as a decision support benchmark. This means, that it simulates a system used to support decision making in business applications. We deemed this an appropriate setting to test distributed query execution on the vertically fragmented data set. The 8 tables in the TPC-H schema are *customer*, *part*, *partsup*, *supplier*, *lineitem*, *orders*, *customer*, *nation* and *region*.

Unfortunately, the TPC-H data generator does not support PostgreSQL and therefore, other tools had to be used to set up the TPC-H benchmark with PostgreSQL. To set up the test database, the HammerDB [22] tool was used with a scale factor of 1. Moreover, the query generator provided by DBT-3 [16] was used to obtain the 22 TPC-H queries conforming with PostgreSQL's standard.

7.3 Settings

The number of tables in the TPC-H database is reasonably small, so that the following artificial scenario is used as the foundation for the tests:

- **Confidentiality Constraints**: The following rules are established for defining the constraints:
  - The name and the account balance of the customers and suppliers are sensitive:
    $c_1 = \{customer.c\_acctbal\}$,
    $c_2 = \{supplier.s\_acctbal\}$
  - The discount given on any order is sensitive:
    $c_3 = \{lineitem.l\_discount\}$
  - A customer's name and its address cannot be placed in the same server fragment:
    $c_4 = \{customer.c\_name, customer.c\_address\}$

  - A customer's name can not be associated with a specific order: $c_5 = \{customer.c\_name, orders.o\_custkey\}$
  - A supplier's name can not be associated with a line item: $c_6 = \{supplier.s\_name, lineitem.l\_suppkey\}$
  - The date of an order can not be associated with the total price: $c_7 = \{orders.o\_odate, orders.o\_totalprice\}$
  - A supplier's name can not be associated with the supplier's cost for a specific part: $c_8 = \{supplier.s\_name, partsupp.ps\_suppkey, partsupp.ps\_supplycost\}$
- **Dependencies**: Moreover, the following dependencies are introduced, concerning personal information about the customers and suppliers:

$$\delta_1 = \{customer.c\_address\} \rightsquigarrow \{customer.c\_name\}$$
$$\delta_2 = \{customer.c\_phone\} \rightsquigarrow \{customer.c\_name\}$$
$$\delta_3 = \{supplier.s\_address\} \rightsquigarrow \{supplier.s\_name\}$$
$$\delta_4 = \{supplier.s\_phone\} \rightsquigarrow \{supplier.s\_name\}$$

- **Visibility Constraints**: As the main purpose of visibility constraints is to speed up the execution of specific queries, a visibility constraint is introduced for each of the 22 queries consisting of all attributes in the query. Therefore, if a visibility constraint is satisfied, the execution of the corresponding query potentially involves a single database server only.

The weights α_1 and α_2 that are also needed for the problem statement are chosen to satisfy the inequality presented in Lemma 2. Finding an optimal solution to this specific instance of the Multi-relational Separation of Duties Problem and setting up the vertically fragmented database takes around two and a half minutes and the resulting fragmentation satisfies 5 of the 22 visibility constraints. Overall, the tables are distributed among 12 table fragments on a total of 3 database servers. One of those is the trusted database server and the remaining two are untrusted.

7.4 Test runs

After the database is set up, the execution time of the 22 queries can be analyzed. For that, each query is executed with the following methods:

1. The original non-fragmented database is queried. To ensure the comparability, the original database is stored separately at the trusted database server.
2. The queries are rewritten by our trusted database client to act on table fragments instead of the original tables. We measured the time for executing (t), the time for rewriting the query (tr), the overall number of table fragments (tf) that are involved in the rewritten query and the slowdown (sd) compared to executing the query on the original database.
3. Instead of rewriting, the queries are cast to specific views set up in the trusted database server to recreate the original tables. For this method, the

execution time (t) is measured, the number of involved table fragments (tf) and the slow down (sd) compared to the execution time of the same query on the original database. The number of involved table fragments is calculated by summing the number of table fragments that were necessary to create every view involved in the query.

4. If a query can be evaluated by a single database server (because it physically stores all the involved attributes), the query is directly cast to this server. For this method, only the execution time (t) is stated because it can be suspected, that their execution time is about the same as for the original database.

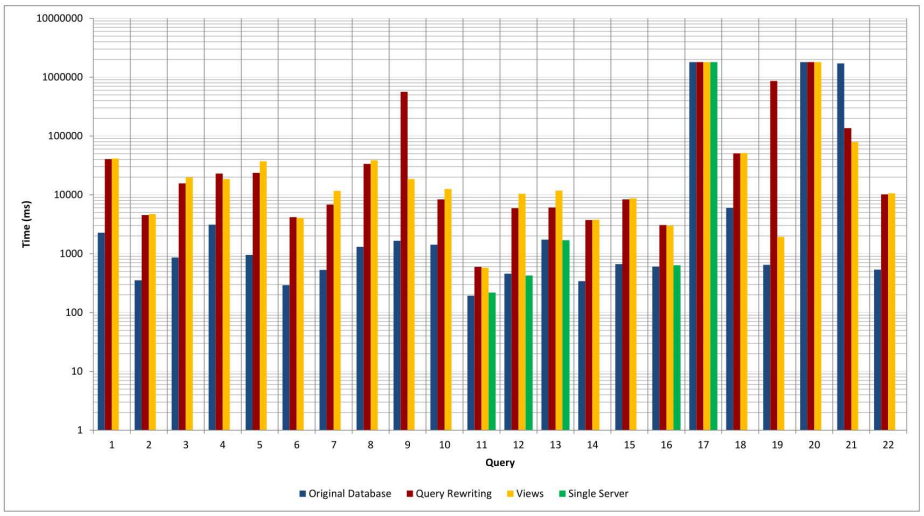

Fig. 15. Runtime results for TPC-H queries

Figure 15 summarizes the results of the test runs while Table 9 shows the exact results. This evaluation shows the major advantage of the separation of duties approach. Because the columns of the tables are outsourced in plaintext, every query can potentially be executed. In particular, we are able to process all queries of the TPC-H benchmark. This is in contrast to approaches using property-preserving encryption: The MONOMI system [31] executes only 19 out of 22 TPC-H queries due to lacking support for views and text pattern matching; according to [31] the CryptDB system [27] executes only four out of the 22 queries.

However, we had to cancel two of the queries, namely Q_{17} and Q_{20}, because the timeout limit (30 minutes) was exceeded. Yet, the reason why these queries take so much time is not related to the vertically fragmented database as the timeout was also reached for the original non-fragmented database. Therefore, it can be concluded this issue is related to the PostgreSQL database engine which

Query	I	II				III			IV
	t (ms)	t (ms)	tr (ms)	tf	sd	t (ms)	tf	sd	t (ms)
Q_1	2267	40413	36	2	17.83×	41180	2	18.16×	n.a.
Q_2	353	4528	396	6	12.83×	4699	6	13.31×	n.a.
Q_3	861	15650	7	4	18.18×	19797	6	22.99×	n.a.
Q_4	3110	22917	12	4	7.37×	18571	4	5.97×	n.a.
Q_5	952	23658	5	7	24.85×	37036	10	38.9×	n.a.
Q_6	291	4167	1	2	14.32×	4039	2	13.88×	n.a.
Q_7	530	6817	14	6	12.86×	11653	9	21.99×	n.a.
Q_8	1305	33453	14	8	25.63×	38584	11	29.57×	n.a.
Q_9	1652	563036	9	7	340.82×	18532	9	11.22×	n.a.
Q_{10}	1417	8340	4	6	5.89×	12547	7	8.85×	n.a.
Q_{11}	193	595	7	3	3.08×	576	5	2.98×	218
Q_{12}	457	5921	4	2	12.96×	10399	4	22.75×	424
Q_{13}	1726	6103	2	2	3.54×	11821	4	6.85×	1696
Q_{14}	341	3721	2	3	10.91×	3765	3	11.04×	n.a.
Q_{15}	663	8368	3	2	12.62×	8685	4	13.1×	n.a.
Q_{16}	603	3054	5	3	5.06×	2983	6	4.95×	634
Q_{17}	timeout	timeout	57	2	n.a.	timeout	3	n.a.	timeout
Q_{18}	5998	50501	11	4	8.42×	51034	6	8.51×	n.a.
Q_{19}	646	859324	5	3	1330.22×	1927	3	2.98×	n.a.
Q_{20}	timeout	timeout	61	5	n.a.	timeout	8	n.a.	n.a.
Q_{21}	1708506	136042	59	7	0.08×	79111	7	0.05×	n.a.
Q_{22}	534	10176	5	4	19.06×	10580	4	19.81×	n.a.

Table 9. Comparison between the different execution methods

cannot find an adequate execution plan for those queries. Notably, [31] report the same problems when running the TCP-H queries: "Queries 17, 20, and 21 cause trouble for the Postgres optimizer: they involve correlated subqueries, which the optimizer is unable to handle efficiently".

As it was suspected, queries Q_{11}, Q_{12}, Q_{13} and Q_{16} that can be evaluated in a reasonable amount of time by a *single* server of the fragmented database can be executed in about the same time as in the non-fragmented database. For these 4 queries, a visibility constraint could be satisfied which perfectly illustrates the benefits of introducing those constraints. Interestingly, query rewriting and using views performed considerably worse for three of those 4 queries (Q_{11}, Q_{12}, Q_{13}). This is especially noticeable because rewriting the query also leads to a situation where the query involves only one database server but this is obviously not detected by PostgreSQL in conjunction with the foreign data wrapper extension postgres_fdw. This observation justifies a prior analysis of the queries as implemented in our distributed database client.

There is one query, Q_{21}, for which the execution time on the fragmented database is lower than the execution time for the non-fragmented database. For this query, the fragmented database probably profited from a better execution strategy that could be established by PostgreSQL due to the query rewriting or

the use of the views. However, we assume that such situations occur very rarely in practice and are caused by PostgreSQL's execution strategy.
An interesting thing to notice is that query rewriting outperformed querying the views 13 times; querying the views was better for only 7 queries. Even more interesting, rewriting the queries performed better in 9 out of 12 times (ignoring the canceled queries) when the number of involved table fragments was lower than for the views. This illustrates the advantage of query rewriting over using views because unnecessary table fragments can be omitted with the former method. The overhead introduced by rewriting the queries is very small for all of the queries compared to the execution time and can therefore be neglected. Consequently, one can conclude that query rewriting is generally the better strategy than using views. However, if for some reason a rewritten takes very long to process, querying the views can potentially lower the execution time. An example for such a situation is query Q_{19}.

8 Conclusion and Future Work

In this article, we extended our separation of duties approach with which confidentiality in cloud databases can be enforced based on vertical fragmentation. Our approach enforces a security policy consisting of confidentiality constraints while at the same time respecting data dependencies, minimizing the amount of external cloud servers (the cardinality of the fragmentation) as well as maximizing the amount of satisfied visibility constraints (the constraints introduced to increase the utility of the resulting fragmentations). An implementation based on the provided theories was presented by translating the separation of duties problem into an integer linear program (ILP) representation and using an off-the-shelf solver to obtain a confidentiality-preserving fragmentation. In addition, we discussed our query rewriting approach, that enables an efficient distributed execution of queries on the fragments.

To show the feasibility of the separation of duties approach, based on the well-known TPC-E database schema the effects of different sizes of input data were evaluated. The evaluation of a TPC-H benchmark showed the major advantage of the separation of duties approach. As the columns of the database are stored in plaintext, it is possible to evaluate any database query, regardless of its complexity. Compared to encryption schemes, there is also no additional resource-intensive workload like decrypting the received data at the database user's site. Therefore, users of cloud databases who potentially run devices with a low computational power, especially benefit from this approach.

Several options for future work arise. Our approach is currently most applicable to situations where the constraint sets remain fixed over time. Studying certain classes of "allowed" modifications of these sets (confidentiality constraints, visibility constraints and dependencies) as well as their influences on security, data distribution and query execution is a major future topic which can be based on [3]. Moreover we plan to provide an in-depth analysis of different classes of integrity constraints similar to [4, 2] as well as considering the query execution

cost as an extra optimization goal. More generally in order to integrate our prior work on property-preserving encryption [34] we aim to analyze the combination of these encryption methods with separation of duties. Lastly it might be worthwhile to analyze the separation of duties approach in non-relational data models [36].

References

1. 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: The Second Biennial Conference on Innovative Data Systems Research (CIDR 2005) (2005)
2. Biskup, J., Preuß, M.: Database fragmentation with encryption: under which semantic constraints and a priori knowledge can two keep a secret? In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 17–32. Springer (2013)
3. Biskup, J., Preuß, M.: Inference-proof data publishing by minimally weakening a database instance. In: International Conference on Information Systems Security. pp. 30–49. Springer (2014)
4. Biskup, J., Preuß, M., Wiese, L.: On the inference-proofness of database fragmentation satisfying confidentiality constraints. In: Information Security Conference. Lecture Notes in Computer Science, vol. 7001, pp. 246–261. Springer (2011)
5. Bollwein, F.: CloudDBSOD Client, http://www.uni-goettingen.de/de/558180.html
6. Bollwein, F., Wiese, L.: Closeness constraints for separation of duties in cloud databases as an optimization problem. In: British International Conference on Databases. pp. 133–145. Springer (2017)
7. Bollwein, F., Wiese, L.: Separation of duties for multiple relations in cloud databases as an optimization problem. In: Proceedings of the 21st International Database Engineering & Applications Symposium. pp. 98–107. ACM (2017)
8. Canim, M., Kantarcioglu, M., Inan, A.: Query optimization in encrypted relational databases by vertical schema partitioning. In: Secure Data Management. pp. 1–16. Springer (2009)
9. Chakravarthy, S., Muthuraj, J., Varadarajan, R., Navathe, S.B.: An objective function for vertically partitioning relations in distributed databases and its analysis. Distributed and parallel databases 2(2), 183–207 (1994)
10. 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: European Symposium on Research in Computer Security. pp. 171–186. Springer (2007)
11. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation design for efficient query execution over sensitive distributed databases. In: ICDCS. pp. 32–39. IEEE Computer Society (2009)
12. 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. Lecture Notes in Computer Science, vol. 5789, pp. 440–455. Springer (2009)
13. 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 (TISSEC) 13(3), 22 (2010)

14. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective data outsourcing for enforcing privacy. Journal of Computer Security 19(3), 531–566 (2011)
15. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: An OBDD approach to enforce confidentiality and visibility constraints in data publishing. Journal of Computer Security 20(5), 463–508 (2012)
16. DBT-3, http://osdldbt.sourceforge.net/
17. De Capitani di Vimercati, S., Erbacher, R.F., Foresti, S., Jajodia, S., Livraga, G., Samarati, P.: Encryption and fragmentation for data confidentiality in the cloud. In: Foundations of Security Analysis and Design VII, pp. 212–243. Springer (2014)
18. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Fragmentation in presence of data dependencies. IEEE Transactions on Dependable and Secure Computing 11(6), 510–523 (2014)
19. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragments and loose associations: Respecting privacy in data publishing. Proceedings of the VLDB Endowment 3(1-2), 1370–1381 (2010)
20. Dwork, C.: Differential privacy: A survey of results. In: International Conference on Theory and Applications of Models of Computation. pp. 1–19. Springer (2008)
21. Göge, C., Waage, T., Homann, D., Wiese, L.: Improving fuzzy searchable encryption with direct bigram embedding. In: International Conference on Trust and Privacy in Digital Business. pp. 115–129. Springer (2017)
22. HammerDB, http://www.hammerdb.com/
23. Homann, D., Göge, C., Wiese, L.: Dynamic similarity search over encrypted data with low leakage. In: International Workshop on Security and Trust Management (in conjunction with ESORICS). pp. 19–35. Springer (2017)
24. Hore, B., Jammalamadaka, R.C., Mehrotra, S.: Flexible anonymization for privacy preserving data publishing: A systematic search based approach. In: Seventh SIAM International Conference on Data Mining. SIAM (2007)
25. Jindal, A., Palatinus, E., Pavlov, V., Dittrich, J.: A comparison of knives for bread slicing. Proceedings of the VLDB Endowment 6(6), 361–372 (2013)
26. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
27. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Processing queries on an encrypted database. Communications of the ACM 55(9), 103–111 (2012)
28. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(05), 557–570 (2002)
29. Transaction Processing Performance Council: TPC-E Benchmark Version 1.14.0, http://www.tpc.org/tpce/
30. Transaction Processing Performance Council: TPC-H Benchmark Version 2.17.1, http://www.tpc.org/tpch/
31. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In: Proceedings of the VLDB Endowment. vol. 6, pp. 289–300. VLDB Endowment (2013)
32. Waage, T., Homann, D., Wiese, L.: Practical application of order-preserving encryption in wide column stores. In: SECRYPT. pp. 352–359. SciTePress (2016)
33. Waage, T., Jhajj, R.S., Wiese, L.: Searchable encryption in Apache Cassandra. In: International Symposium on Foundations and Practice of Security. pp. 286–293. Springer (2015)

34. Waage, T., Wiese, L.: Property preserving encryption in nosql wide column stores. In: Cloud and Trusted Computing (OnTheMove Federated Conferences). pp. 3–21. Springer (2017)
35. Wiese, L.: Horizontal fragmentation for data outsourcing with formula-based confidentiality constraints. In: IWSEC. Lecture Notes in Computer Science, vol. 6434, pp. 101–116. Springer (2010)
36. Wiese, L.: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases. DeGruyter/Oldenbourg (2015)
37. Xiao, Y., Xiong, L., Yuan, C.: Differentially private data release through multidimensional partitioning. In: 7th VLDB Workshop on Secure Data Management. Lecture Notes in Computer Science, vol. 6358, pp. 150–168. Springer (2010)
38. Zakerzadeh, H., Aggarwal, C.C., Barker, K.: Managing dimensionality in data privacy anonymization. Knowledge and Information Systems 49(1), 341–373 (2016)
39. Zhang, J., Xiao, X., Xie, X.: Privtree: A differentially private algorithm for hierarchical decompositions. In: Proceedings of the 2016 International Conference on Management of Data. pp. 155–170. ACM (2016)