Encryption and fragmentation for data confidentiality in the cloud

Sabrina De Capitani di Vimercati[1], Robert F. Erbacher[2], Sara Foresti[1], Sushil Jajodia[3], Giovanni Livraga[1], and Pierangela Samarati[1]

[1] Dipartimento di Informatica – Università degli Studi di Milano
Via Bramante 65, 26013 Crema, Italy
firstname.lastname@unimi.it
[2] U.S. Army Research Laboratory, USA
2800 Powder Mill Road, Adelphi, MD 20783, USA
robert.f.erbacher.civ@mail.mil
[3] Center for Secure Information Systems – George Mason University
4400 University Drive, Fairfax, VA 22030-4422, USA
jajodia@gmu.edu

Abstract. Cloud computing has emerged as a successful paradigm allowing individual users and well as companies to resort to external providers for storing/processing data or making them available to others. Together with the many benefits, cloud computing introduces however new security and privacy risks. A major issue is due to the fact that the data owner, storing data at external providers, loses control over them, leaving them potentially exposed to improper access, use, or dissemination.
In this chapter, we consider the problem of protecting confidentiality of sensitive information when relying on external cloud providers for storing and processing data. We introduce confidentiality requirements and then illustrate encryption and data fragmentation as possible protection techniques. In particular, we discuss different approaches that have been proposed using encryption (with indexing) and fragmentation, either by themselves or in combination, to satisfy confidentiality requirements.

1 Introduction

Cloud computing has brought enormous benefits to individual users as well as companies, enabling them to enjoy convenient and flexible availability of on demand storage and computational resources for storing, processing and share data with others. While these advantages are appealing, the price to pay for them is a loss of control of the data owners on their data, whose confidentiality and integrity could then be put at risk [20,32]. Security issues may vary depending on the considered cloud scenario. In fact, the term cloud refers to a variety of distributed computing environments, which differ in the architectural or trust assumptions. Specifically, different deployment models can be identified [36], ranging from *private cloud*, which operates for a single organization

and the infrastructures and services are maintained on a private network, to a *public cloud*, where the cloud infrastructure is owned by a cloud provider that offers its services to everybody. Ownerships and operation models between these two extremes are also possible, such as in a *community cloud*, where the cloud infrastructure is shared among a set of organizations with similar needs, and a *hybrid cloud*, where an organization with a private cloud wants to use it in conjunction with a public or community cloud for a given purpose (e.g., critical applications and data are managed in the private cloud while other less critical applications can be managed in a public cloud). In all models above, the consideration of (not fully trusted/trustworthy) providers introduces potential risks on the protection of data that are stored or processed by external providers. In particular, while providers could typically be assumed to be trustworthy with respect to the proper management of the data, they might not be trusted for data confidentiality. In other words, data should be protected from the providers themselves (considered *honest-but-curious*) that, while providing data storage, management, and processing, should not be authorized to know the actual data content.

In this chapter, we address the problem of guaranteeing data confidentiality when relying on external cloud providers for storing and processing data and illustrate possible solutions for it. In particular, a natural solution for protecting data confidentiality is *encryption*: data are protected by applying an encryption layer wrapping them before outsourcing them to external cloud providers. However, while effective, encryption makes query execution more complex. In fact, the external provider cannot decrypt the data for query execution, and must execute queries directly on encrypted data (not always applicable in practice) or rely on indexing information that can be associated with encrypted data. An additional/alternative solution is data *fragmentation*: when what is sensitive is the association among data (rather than the individual data themselves), confidentiality can be provided by storing different chunks of the data in separate non-linkable fragments.

The remainder of this chapter is organized as follows. Section 2 introduces the protection requirements to be enforced as a set of confidentiality constraints, and describes encryption and fragmentation as basic techniques that can be used to preserve the confidentiality of stored data. It also introduces the different data protection paradigms given by the disjoint or combined application of encryption and fragmentation, which are illustrated in more details in subsequent sections. Section 3 illustrates protection via data encryption (and indexing). Section 4 illustrates an approach departing from encryption in favor of data fragmentation whenever possible and assuming two data fragments and the availability of two independent and non-communicating providers for storing them. Section 5 illustrates a similar approach assuming an arbitrary number of non-linkable data fragments, which can be stored at an arbitrary number (including one) of providers on which no specific assumption is required. Section 6 illustrates an approach completely departing from encryption relying instead only on frag-

mentation and assuming the owner's involvement in storing (and processing) a limited amount of data. Finally, Section 7 concludes the chapter.

2 Protection Requirements and Techniques

In this section, we first discuss confidentiality requirements that may need to be satisfied when moving the data to the cloud (Section 2.1), and then describe the protection techniques that can adopted for their enforcement (Section 2.2). Finally, we illustrate the data protection paradigms resulting from different combinations of these protection techniques (Section 2.3).

2.1 Confidentiality constraints

Protection requirements express what is sensitive and should be therefore maintained confidential when storing data at external providers. For simplicity and concreteness, most existing proposals assume data to be organized as a relation r over relational schema $R(A_1, \ldots, A_n)$, where A_i, $i = 1, \ldots, n$, are the different attributes of the relation, with the note that the proposed protection techniques could however be applied to different data models. Similarly, they assume protection requirements to be defined at the schema level, meaning at the level of attributes (in contrast to specific attribute values). This assumption simplifies the management and the application of the protection techniques, ensuring the applicability of the solutions.

Operating at the schema level, we can distinguish the following two kinds of confidentiality requirements that can apply to the data, corresponding to the fact that a given attribute is sensitive or that the association among some attributes is sensitive.

- *Sensitive attributes.* Some attributes are sensitive and their values should be maintained confidential. Simple examples of such attributes are SSN, credit card numbers, emails or telephone numbers and similar attributes whose values should not be released.
- *Sensitive associations.* In some cases, what is sensitive is the association among attributes values rather than the values of an attribute. For instance, the names of patients in a hospital may be considered not sensitive, and so the diseases treated by the hospital; however the specific association between individual patients and their illnesses is sensitive and should be maintained confidential.

A simple, yet conveniently expressive, way to capture the confidentiality requirements of sensitive attributes/associations is the specification of confidentiality constraints as set of attributes whose joint visibility should be avoided [1]. Singleton sets correspond to sensitive attributes; non-singleton sets correspond to sensitive associations.

PATIENTS

SSN	Name	Race	Job	Disease	Treatment	Ins
123-45-6789	Alice	white	teacher	flu	paracetamol	160
234-56-7890	Bob	while	farmer	asthma	bronchodilators	100
345-67-8901	Carol	asian	nurse	gastritis	antacids	100
456-78-9012	David	black	lawyer	angina	nitroglycerin	200
567-89-0123	Eric	black	secretary	flu	aspirin	100
678-90-1234	Fred	asian	lawyer	diabetes	insulin	180

(a)

$c_1 = \{\texttt{SSN}\}$
$c_2 = \{\texttt{Name}, \texttt{Disease}\}$
$c_3 = \{\texttt{Name}, \texttt{Ins}\}$
$c_4 = \{\texttt{Disease}, \texttt{Ins}\}$
$c_5 = \{\texttt{Race}, \texttt{Job}, \texttt{Ins}\}$

(b)

Fig. 1. An example of a relation (a) and of confidentiality constraints over it (b)

Definition 1 (Confidentiality constraint). *Let $R(A_1, \ldots, A_n)$ be a relation schema. A confidentiality constraint c over R is a subset of attributes in R (i.e., $c \subseteq \{A_1, \ldots, A_n\}$)*

As an example, consider relation PATIENTS in Figure 1(a), reporting the information about hospitalized patients. Figure 1(b) illustrates an example of confidentiality constraints over it stating that:

- c_1: the Social Security Numbers of the patients are sensitive and should be maintained confidential (sensitive attribute);
- c_2, c_3: the disease suffered from a patient and the medical insurance she pays are sensitive and should be maintained confidential (sensitive associations);
- c_4: the association between the disease of a patient and the medical insurance she pays is sensitive (sensitive association);
- c_5: the association among the race of a patient, her job, and the insurance she pays is confidential (sensitive association).

Note that the protection of a confidentiality constraint c_i implies the protection of any confidentiality constraint c_j such that $c_i \subset c_j$ (if observers do not have visibility of the attribute/association c_i they clearly do not have visibility of the association including it); making the consideration of c_j redundant. A set $\mathcal{C}$ of confidentiality constraints over R is *well-defined* if it does not include *redundant* constraints, that is, $\forall c_i, c_j \in \mathcal{C}$, $i \neq j$: $c_i \not\subset c_j$. The set of constraints in Figure 1(b) is well-defined.

2.2 Encryption and fragmentation

Two natural protection techniques that have been proposed for satisfying confidentiality requirements are *encryption* and *fragmentation*.

Encryption consists in encrypting the data before outsourcing them to external providers so to make them intelligible only to users holding the decryption

keys, and protecting them from unauthorized eyes (including the provider itself). Although in principle both symmetric and asymmetric encryption schemas can be adopted, for performance reasons, most proposals assume the adoption of symmetric encryption. Encryption could be enforced at different levels of granularity: table, column, tuple, and individual cell. Encrypting at the level of table implies that the whole relation needs to be returned to the client for access, requiring heavy communication and leaving the whole query processing work to the client. Such a drawback is also present in case of encryption at the level of column as the only operation that the provider could perform is projection, with the whole column of interest for a query being always returned. On the other hand, encrypting at the level of individual cells would introduce many encryption/decryption operations and issues related to possible inferences on the encrypted data. Encrypting at the level of tuple appears then the preferable option, providing some ability for fine-grained retrieval (returning only a subset of the tuples) while not requiring too many encryption/decryption operations.

Since the provider is not trusted for confidentiality, encrypted data cannot be decrypted for query execution. Queries need therefore to be evaluated on the encrypted data themselves. There are typically two lines of approaches for providing this functionality: performing queries directly on encrypted data, possibly with the use of specific cryptographic techniques (e.g., [6,13,25,34]), or attaching to the encrypted data some metadata representing indexes that are then exploited for query execution (e.g., [5,14,22,26]). These approaches however support evaluation of only specific kinds of queries.

Fragmentation consists in splitting the attributes of a relation R producing different vertical views (fragments) in such a way that these views stored at external providers do not violate confidentiality requirements (neither directly nor indirectly). Intuitively, fragmentation protects the sensitive association represented by an association constraint c when the attributes in c do not appear all in the same (publicly available) fragment, and fragments cannot be joined by non authorized users. Note that singleton constraints are correctly enforced only when the corresponding attributes do not appear in any fragment that is stored at a cloud provider. In this chapter, we illustrate fragmentation solutions assuming attributes to be independent. Fragmentation can however take into account also the case of possible correlations among attributes (which could introduce inferences or enable linking) [16].

2.3 Data protection paradigms

Different approaches have been proposed for protecting confidentiality of data stored at external providers by applying encryption and fragmentation, by themselves or in combination. We distinguish four different protection paradigms that have been proposed.

– *Encryption/indexing*. Data are encrypted before being outsourced to external providers, with encryption typically applied at the level of tuple. Encryption does not distinguish between attributes or association constraints,

applying instead the wrapping protection layer to all the attributes in a tuple. For query purposes, the encrypted data are associated with some metadata (indexes) that can be used by the cloud provider for executing queries.

- *Two can keep a secret*. It assumes the availability of two independent, non-communicating, providers each storing a portion of the data. Whenever possible, sensitive associations are protected by partitioning the involved attributes among the two providers (any way would do, as long as none of the two providers has complete visibility of all the attributes in a sensitive association). Sensitive attributes are always encrypted. Other attributes may be stored in encrypted form whenever storing them in the clear at any of the two providers would violate at least one confidentiality constraint. The two fragments have a key attribute in common, making them joinable by the owner and by authorized users, who are the only parties who have access to both providers.
- *Multiple fragments*. It employs encryption for protecting sensitive attributes and fragmentation for protecting sensitive associations. It does not make assumptions on the nature/number of providers and on the number of fragments. Employing an arbitrary number of fragments allows sensitive associations to always be satisfied with fragmentation. Fragments are complete (all attributes are stored in each fragment in either encrypted or plaintext form) and not linkable (they have no attribute in common). Being fragments unlinkable, there is no need of assuming absence of communication between the providers.
- *Keep a few*. It assumes the involvement of a trusted party (typically the owner) for storing, and hence participating in the processing of, a limited amount of data. No encryption is applied. Sensitive attributes are stored at the owner side. Sensitive associations are protected by storing at least one of their attributes at the owner side (trying to minimize storage/computation required to the owner).

In the following sections, we describe more in details these four data protection paradigms. For the encryption/indexing paradigm, we will present the data model and describe how to execute queries directly on the encrypted data. For the fragmentation-based paradigms, we will present: *i)* the fragmentation model; *ii)* the metrics for evaluating the quality of a fragmentation; *iii)* the algorithms developed for computing an optimal fragmentation; and *iv)* the techniques to efficiently evaluate queries on the fragmentation.

3 Encryption and indexing

We first describe how data confidentiality can be guaranteed by encrypting the data before storing them in the cloud [14,26] (Figure 2), and then illustrate how indexes can be defined and adopted for supporting the execution of queries.

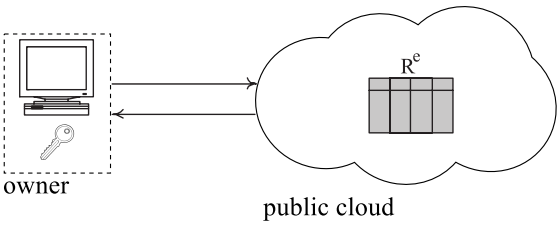

Fig. 2. Encryption and indexing

PATIENTS^e

tid	enc	$\mathbf{I}_r$	$\mathbf{I}_j$	$\mathbf{I}_d$	$\mathbf{I}_i$
1	4tBf	α	δ	ζ	σ
2	lkG7	α	δ	η	ρ
3	wF4t	β	ϵ	θ	ρ
4	m;Oi	γ	ϵ	κ	σ
5	n:8u	γ	δ	λ	ρ
6	xF-g	β	ϵ	μ	σ

Fig. 3. An example of encrypted and indexed version of relation PATIENTS in Figure 1(a)

Encryption model A relation r, defined over schema $R(A_1, \ldots, A_n)$, is represented at the cloud provider as an encrypted and indexed relation r^e, defined over schema $R^e(\underline{tid}, enc, I_i, \ldots, I_j)$, where:

- *tid* is a randomly generated tuple identifier;
- *enc* is the encrypted tuple;
- $\{I_i, \ldots, I_j\}$ is the set of indexes defined over attributes $\{A_i, \ldots, A_j\} \subseteq R$.

Each tuple t in r is represented by an encrypted tuple t^e in r^e, where: $t^e[tid]$ is a randomly generated value; $t^e[enc]=Enc(t,k)$, with *Enc* a symmetric encryption function with key k; and $t^e[I_l]=\iota_l(t[A_l])$, with ι_l an index function defined for attribute A_l. Note that not all the attributes in R are associated with an index in the corresponding encrypted relation R^e. Typically, indexes are defined only for those attributes on which conditions need to be evaluated in query execution. Figure 3 illustrates the encrypted version of relation PATIENTS in Figure 1(a), with indexes over attributes `Race`, `Job`, `Disease`, and `Ins`. In the figure, for simplicity, index values are represented with Greek letters.

Depending on how the index function ι maps plaintext values into the corresponding index values, most of the existing indexing techniques can be classified as follows [22].

- *Direct index*: maps each value in the attribute domain to a different index value and viceversa. Encryption-based indexes (e.g., [14]) represent an example of direct index. In fact, the index function maps plaintext value $t[A]$ to index value $\iota(t[A])=E_k(t[A])$, for each tuple t in r. For instance, index I_r in relation PATIENTS^e in Figure 3 is a direct index over attribute `Race` of relation PATIENTS in Figure 1(a).

- *Bucket-based index*: maps different values in the attribute domain to the same index value (i.e., generates collisions), but each value in the attribute domain is mapped to one index value only. Partition-based and hash-based indexes are examples of bucket-based indexes. Partition-based indexes (e.g., [26]) split the domain D of attribute A into non-overlapping subsets of contiguous values and associate a label with each of them. The index value representing $t[A]$, for each tuple t in r, is the label of the partition to which $t[A]$ belongs. For instance, index I_i in relation PATIENTS^e in Figure 3 is a partition-based index over attribute `Ins` of relation PATIENTS in Figure 1(a), where the domain has been partitioned in two intervals: [100, 150] with label ρ, and [151,200] with label σ. Hash-based indexes (e.g., [14]) instead adopt a secure hash function h that generates collisions. Hence, the index value representing $t[A]$ is computed as $h(t[A])$, for each tuple t in r. For instance, index I_j in relation PATIENTS^e in Figure 3 is a hash-based index over attribute `Job` of relation PATIENTS in Figure 1(a), where the hash function is defined as follows: h(teacher)=h(farmer)=h(secretary)=δ and h(nurse)=h(lawyer)=ϵ.
- *Flattened index*: maps each value in the attribute domain to a set of index values, in such a way that all the index values have the same number of occurrences (flattening). Each index value however represents one value in the attribute domain only. An example of flattened index applies direct encryption to the values in the attribute domain and a post-processing to flatten the distribution of index values. For instance, index I_d in relation PATIENTS^e in Figure 3 represents a flattened index over attribute `Disease` of relation PATIENTS in Figure 1(a), where each index value has one occurrence.

Besides these approaches, indexing techniques have been proposed for supporting the evaluation of specific conditions and SQL clauses [21]. As an example, solutions that exploit homomorphic encryption have been developed to support aggregate functions and the basic arithmetic operators (e.g., [24,27]). Techniques based on the Order Preserving Encryption schema have been instead studied to support range conditions and ordering (e.g., [2,35]). A different class of indexing techniques rely on the definition of specific data structures (e.g., B+-tree) to support the evaluation at the provider-side of specific operations. These indexes are however not represented as attributes of the encrypted relation, but they translate into additional relations stored together with r^e at the cloud provider [14].

Query evaluation Since moving data to the cloud should be transparent for final users, they formulate their queries over the original relation schema. These queries are then translated into equivalent queries operating on the encrypted and indexed relation r^e. The translation of a query q operating on the original relation into an equivalent set of queries exploiting indexes depends on the indexing techniques adopted by the data owner.

Consider, for simplicity, a query q of the form "SELECT *Att* FROM R WHERE *Cond*", with *Att* a set of attributes in R and $Cond = \bigwedge_i cond_i$ is the conjunction of equality conditions of the form $A=v$, with $A \in R$ and v a value in its domain.

Original query	Translated queries
$q :=$ SELECT Att FROM R WHERE $Cond$	$q_p :=$ SELECT tid, enc FROM R^e WHERE ${Cond_p}^e$ AND ${Cond_{pu}}^e$ $q_u :=$ SELECT Att FROM $Decrypt(R_p.enc,k)$ WHERE $Cond_u$ AND $Cond_{pu}$
$q :=$ SELECT `Name` FROM `Patients` WHERE `Race`='white' AND `Job`='teacher' AND `Treatment`='paracetamol'	$q_p :=$ SELECT `tid`, `enc` FROM `Patients`^{e} WHERE $\mathtt{I}_r=\alpha$ AND $\mathtt{I}_j=\delta$ $q_u :=$ SELECT `Name` FROM $Decrypt(R_p.enc,k)$ WHERE `Job`='teacher' AND `Treatment`='paracetamol'

Fig. 4. An example of query translation in the encryption and indexing scenario

To partially delegate the query evaluation to the cloud provider storing r^e, q is translated into two queries: q_p executed by the provider and q_u executed by the user. Query q_p contains only the equality conditions in the WHERE clause of q that operate on indexes. Query q_u operates on the result of q_p and contains all the other conditions. To translate q into an equivalent pair of queries $\{q_p,q_u\}$, $Cond$ is first split in sub-conditions $Cond_p$, $Cond_u$, and $Cond_{pu}$ as follows:

- $Cond_p$ is the conjunction of conditions $cond$ in $Cond$ involving attributes that are represented by an index in r^e that fully support equality conditions (e.g., direct and flattened indexes);
- $Cond_u$ is the conjunction of conditions $cond$ in $Cond$ that involve attributes that are not represented by an index in r^e;
- $Cond_{pu}$ is the conjunction of conditions $cond$ in $Cond$ that involve attributes that are represented by any index in r^e that only partially supports the evaluation of equality conditions (e.g., bucket-based indexes, due to collisions).

For instance, consider the encrypted and indexed version of relation PATIENTS in Figure 1(a) reported in Figure 3, and query "SELECT `Name` FROM `Patients` WHERE `Race`='white' AND `Job`='teacher' AND `Treatment`='paracetamol'". In this case, $Cond_p$={`Race`='white'}, $Cond_u$={`Treatment`='paracetamol'}, and $Cond_{pu}$={`Job`='teacher'}.

After conditions in $Cond$ have been classified in $Cond_p$, $Cond_u$, and $Cond_{pu}$, query q is translated in q_p and q_u, as illustrated in Figure 4. Query q_p, evaluated by the provider, operates on r^e and evaluates the conditions in $Cond_p$ and in $Cond_{pu}$, properly translated to operate on indexes. That is, each condition $(A_i{=}v)$ is represented in q_p by condition $(I \text{ IN } \iota(v))$, with I the index defined over A and ι the corresponding index function. When the user receives the result R_p of query q_p, it decrypts attribute enc and evaluates, on the resulting tuples, query q_u. Query q_u evaluates conditions in $Cond_u$ and $Cond_{pu}$

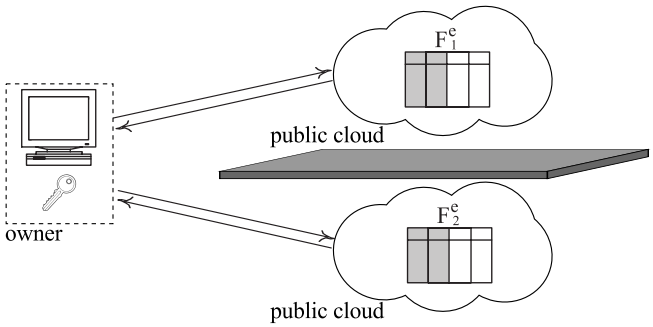

Fig. 5. Two can keep a secret

and projects the attributes *Att*. Consider, as an example, the encrypted and indexed version of relation PATIENTS in Figure 1(a) reported in Figure 3, and query "SELECT `Name` FROM `Patients` WHERE `Race`='white' AND `Job`='teacher' AND `Treatment`='paracetamol'". Figure 4 illustrates the translation of q in the corresponding sub-queries operating at the provider (i.e., q_p) and at the user (i.e., q_u) sides.

4 Two can keep a secret

We present a solution based on encryption and fragmentation where data are split into two fragments, with each fragment stored at a different provider (Figure 5). The two providers are assumed to be non-communicating [1].

Fragmentation model The satisfaction of confidentiality constraints is guaranteed by proper combination of vertical fragmentation and encryption, and relies on the assumption that the two cloud providers storing fragments do not communicate with each other (see Figure 5). Note that in the original proposal [1] encryption is considered as one of the techniques that can be used for encoding (i.e., obfuscating) attributes. Given an attribute A, the encoding of A consists in splitting its value in two (or more) attributes, say A^i and A^j, whose combined knowledge is necessary to reconstruct A (i.e., $A = A^i \otimes A^j$ with $\otimes$ a non-invertible composition operator). Encoding an attribute using encryption means therefore that A^i contains the ciphertext, A^j contains the encryption key, and $\otimes$ is the encryption function adopted by the data owner. For the sake of readability, in the remainder of this section we will consider encryption as the specific technique adopted to enforce encoding.

According to the proposal in [1], the original relation r is fragmented generating a fragmentation $\mathcal{F}$=$\{F_1,F_2,E\}$, where F_1 and F_2 are two fragments that are stored at two providers and E is the set of encrypted attributes. Singleton constraints are satisfied by encrypting sensitive attributes. Association constraints are satisfied by splitting the involved attributes between the two providers. Since

relation r can be split in two fragments only, it may happen that an attribute cannot be stored at any of the two providers without violating a confidentiality constraint. In this case, the confidentiality constraint can be satisfied by encrypting one (or more) of its attributes. A fragmentation $\mathcal{F}$=$\{F_1$,F_2,$E\}$ is *correct* if it satisfies all the confidentiality constraints defined by the data owner, as formally stated below.

Definition 2 (Correct Fragmentation). *Let $R(A_1, \ldots, A_n)$ be a relation schema and $\mathcal{C}$ be a set of confidentiality constraints over it. A fragmentation $\mathcal{F} = \{F_1, F_2, E\}$ is* correct *iff:*

- $\forall c \in \mathcal{C}$: $c \not\subseteq F_1$, $c \not\subseteq F_2$ (*confidentiality*),
- $F_1 \cup F_2 \cup E = R$ (*completeness*).

The first condition requires that neither F_1 nor F_2 store all the attributes in a confidentiality constraint in plaintext. Since the two fragments are stored at different providers, and these providers do not communicate with each other, sensitive associations as well as sensitive attribute values cannot be reconstructed by non authorized users. The second condition instead demands that the fragments store (either plaintext or encrypted) all the attributes in the original relation. This guarantees that the content of the original relation can always be reconstructed starting from $\mathcal{F}$. For instance, a correct fragmentation $\mathcal{F}$ of relation PATIENTS in Figure 1(a) with respect to the confidentiality constraints in Figure 1(b) is $\mathcal{F}$=$\{F_1$,F_2,$E\}$, with F_1=$\{$`Name`,`Race`,`Job`$\}$, F_2=$\{$`Disease`,`Treatment`$\}$, and E=$\{$`SSN`,`Ins`$\}$.

At the physical level, fragments F_1 and F_2 are represented by *physical fragments* F_1^e and F_2^e, respectively. Each physical fragment F_i^e stores the attributes in F_i in plaintext, and all the attributes in E encrypted. The two physical fragments representing relation r must have a common attribute, to allow authorized users to correctly reconstruct the content of r (lossless join property). Therefore, physical fragment F_i^e representing fragment F_i has schema $F_i^e(\underline{tid}, A_{i_1}, \ldots, A_{i_n}, A^i_{e_1}, \ldots, A^i_{e_m})$, where:

- *tid* is a randomly generated tuple identifier;
- $\{A_{i_1}, \ldots, A_{i_n}\}$ is the set of attributes composing fragment F_i;
- $\{A^i_{e_1}, \ldots, A^i_{e_m}\}$ is the set of attributes resulting from the encryption of the attributes in E=$\{A_{e_1}, \ldots, A_{e_m}\}$, that is, for each A_{e_i} in E, either $A^1_{e_i}$ represents encrypted attribute A_{e_i} and $A^2_{e_i}$ represents the corresponding encryption key, or viceversa.

Each tuple t in r is represented by a tuple t_1^e in F_1^e and a tuple t_2^e in F_2^e, where: $t_1^e[tid]$=$t_2^e[tid]$ is a randomly generated value; $t_1^e[A]$=$t[A]$, $\forall A \in F_1$ and $t_2^e[A]$=$t[A]$, $\forall A \in F_2$; and attributes $t_1^e[A^1]$, $t_2[A^2]$ are the encrypted version and the encryption key of attribute $t[A]$, $\forall A \in E$ (i.e., $Enc(t[A], t_1^e[A^1]) = t_2^e[A^2]$ or $Enc(t[A], t_2^e[A^2]) = t_1^e[A^1]$).

Figure 6 illustrates the physical fragments representing fragmentation $\mathcal{F}$=$\{F_1$,F_2,$E\}$, with F_1=$\{$`Name`,`Race`,`Job`$\}$, F_2=$\{$`Disease`,`Treatment`$\}$, and

F_1^e

tid	Name	Race	Job	SSN^1	Ins^1
1	Alice	white	teacher	Enc(123-45-6789,$k^1_{\texttt{SSN}}$)	Enc(150,$k^1_{\texttt{Ins}}$)
2	Bob	while	farmer	Enc(234-56-7890,$k^2_{\texttt{SSN}}$)	Enc(100,$k^2_{\texttt{Ins}}$)
3	Carol	asian	nurse	Enc(345-67-8901,$k^3_{\texttt{SSN}}$)	Enc(100,$k^3_{\texttt{Ins}}$)
4	David	black	lawyer	Enc(456-78-9012,$k^4_{\texttt{SSN}}$)	Enc(200,$k^4_{\texttt{Ins}}$)
5	Eric	black	secretary	Enc(567-89-0123,$k^5_{\texttt{SSN}}$)	Enc(100,$k^5_{\texttt{Ins}}$)
6	Fred	asian	lawyer	Enc(678-90-1234,$k^6_{\texttt{SSN}}$)	Enc(180,$k^6_{\texttt{Ins}}$)

F_2^e

tid	Disease	Treatment	SSN^2	Ins^2
1	flu	paracetamol	$k^1_{\texttt{SSN}}$	$k^1_{\texttt{Ins}}$
2	asthma	bronchodilators	$k^2_{\texttt{SSN}}$	$k^2_{\texttt{Ins}}$
3	gastritis	antacids	$k^3_{\texttt{SSN}}$	$k^3_{\texttt{Ins}}$
4	angina	nitroglycerin	$k^4_{\texttt{SSN}}$	$k^4_{\texttt{Ins}}$
5	flu	aspirin	$k^5_{\texttt{SSN}}$	$k^5_{\texttt{Ins}}$
6	diabetes	insulin	$k^6_{\texttt{SSN}}$	$k^6_{\texttt{Ins}}$

Fig. 6. An example of a correct fragmentation of relation PATIENTS in Figure 1(a) in the two can keep a secret scenario

E={SSN,Ins} of relation PATIENTS in Figure 1(a). In this example, for simplicity, we assume that F_1^e stores the encrypted attribute values and F_2^e stores the corresponding encryption keys for all the tuples in r and for both attributes SSN and Ins.

Fragmentation metrics Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, the data owner needs to compute a correct fragmentation. However, there may exist different fragmentations that satisfy all the constraints. As a simple example, fragmentation $\mathcal{F}$={F_1,F_2,E} with E=R and F_1=F_2=$\emptyset$ is clearly correct but undesirable, since no query can be evaluated by the providers storing F_1^e and F_2^e. Aiming at leaving as much computational effort as possible to the cloud providers, it is then necessary to define a metric to measure the *quality* of a fragmentation in terms of the query overhead required to users for evaluating their queries over the fragmentation $\mathcal{F}$. The metric proposed in [1] is based on the knowledge of the query workload $\mathcal{Q}$ (i.e., a set of representative queries that are expected to be frequently executed) characterizing the system. In fact, the query workload describes how frequently attributes appear together in queries, and then permits to estimate the computational overhead that a fragmentation that splits these attributes may cause to users. To assess the quality of a fragmentation, the query workload is modeled as an *affinity matrix*, which is a symmetric matrix with a row and a column for each attribute in R, and where each cell $M[A_i,A_j]$=$M[A_j,A_i]$ ($i \neq j$), represents the cost (i.e., the computation overhead for users) of having attributes A_i and A_j stored in different fragments. Each cell $M[A,A]$ (i.e., cells along the diagonal) instead represents the cost of having attribute A encrypted. For instance, the affinity matrix in Figure 7 states that the cost of having attributes Name and Disease stored in two different fragments is $M[\texttt{Name},\texttt{Disease}] = 10$, and that of encrypting attribute Ins is $M[\texttt{Ins},\texttt{Ins}] = 15$. The cost of a fragmentation $\mathcal{F}$ is computed by summing the costs of the attributes encrypted in $\mathcal{F}$, and the costs of the pairs

	SSN	Name	Race	Job	Disease	Treatment	Ins
SSN	10	20	22	18	17	25	10
Name		30	12	25	10	15	40
Race			20	18	32	40	10
Job				10	14	23	17
Disease					10	30	40
Treatment						20	40
Ins							15

Fig. 7. An example of affinity matrix

of attributes not stored together in a fragment in $\mathcal{F}$. Formally, the cost of a fragmentation $\mathcal{F}$=$\{F_1,F_2,E\}$ is defined as:

$$\sum_{A_i \in F_1, A_j \in F_2} M[A_i, A_j] + \sum_{A_i \in E} M[A_i, A_i]$$

As an example, consider relation PATIENTS in Figure 1(a), the fragmentation in Figure 6, and the affinity matrix in Figure 7 (since the matrix is symmetric, we report the values only for the cells in the upper half of the matrix). The quality of $\mathcal{F}$ is computed as: $M[\texttt{Name}, \texttt{Disease}] + M[\texttt{Name}, \texttt{Treatment}] + M[\texttt{Race}, \texttt{Disease}] + M[\texttt{Race}, \texttt{Treatment}] + M[\texttt{Job}, \texttt{Disease}] + M[\texttt{Job}, \texttt{Treatment}] + M[\texttt{SSN}, \texttt{SSN}] + M[\texttt{Ins}, \texttt{Ins}] = 10 + 15 + 32 + 40 + 14 + 23 + 10 + 15 = 159$.

Computing an optimal fragmentation The problem of computing a fragmentation that minimizes the cost of query evaluation is NP-hard (the minimum hypergraph coloring problem reduces to it in polynomial time [1]). Hence, in [1] the authors propose to adopt an heuristic approach to compute a good, although non optimal, solution. The proposed solution is based on a graph modeling of the fragmentation problem, where each attribute in R is represented as a vertex in a *complete graph* G whose edges and vertices are weighted according to M (i.e., *weight*(A)=$M[A,A]$ and *weight*(A_i,A_j)=$M[A_i,A_j]$). The graph has an additional set H of hyperarcs, modeling the confidentiality constraints in $\mathcal{C}$. The proposed heuristic combines two approximation techniques, traditionally used to find a good solution to the following well known hard problems.

- *Min-Cut*. Assuming that $\mathcal{C}$ is empty, the problem of computing an optimal fragmentation can be translated into the problem of computing a minimum cut for G. A minimum cut for a graph G is a partitioning of the set of vertices in G in two subsets, V_1 and V_2, that minimizes the weight of the edges with one vertex in V_1 and vertex in V_2. Intuitively, the heuristic approaches proposed for the Min-Cut problem can be used to compute different cuts that are nearly minimal, and then we can choose the one that satisfies the highest number of confidentiality constraints.
- *Weighted Set Cover*. If we do not consider the cost of splitting attributes between F_1 and F_2, the problem of computing a correct fragmentation can be translated into the minimum set cover problem. Intuitively, each confidentiality constraint is a set whose elements are the attributes composing

it. The weight of each attribute A is $M[A,A]$ and the minimum set cover is the set C' of attributes with minimum weight that includes (at least) one attribute for each constraint. Confidentiality constraints can be satisfied by encrypting all the attributes in C'.

By combining these two approaches, it is possible to compute a good fragmentation in polynomial time. In fact, the Min-Cut heuristic algorithm guarantees to compute a good split of the attributes between F_1 and F_2, while the weighted set cover guarantees constraint satisfaction. The corresponding heuristic algorithm works as follows. First, it computes a minimum set cover E through a greedy strategy, to guarantee that all the constraints are satisfied by encrypting the attributes in E. Then, it computes a minimum cut for the attributes in $R \backslash E$, to split them between F_1 and F_2 minimizing the cost of the fragmentation. Finally, for each attribute A in E, it moves A to F_1 (F_2, respectively) if no confidentiality constraint is violated.

Query evaluation A query q formulated over the original relation r is translated into a set of queries operating over the two fragments stored at the two cloud providers. A naive solution would download from the two providers both F_1^e and F_2^e, and locally evaluate q at the user side on the joined fragments. However, this solution would not be acceptable due to the high computational and communication overhead for users. The translation of original queries to operate on fragments should then limit the computational overhead for the user (i.e., moving as much as possible the query evaluation process to the providers).

Consider, for simplicity, a query q of the form "SELECT *Att* FROM R WHERE *Cond*", with *Att* a set of attributes in R and $Cond = \bigwedge_i cond_i$ a conjunction of basic conditions of the form (A_i *op* v), (A_i *op* A_j), or (A_i IN $\{v_i, \ldots, v_k\}$), where $A_i, A_j \in R$, $\{v, v_1, \ldots, v_k\}$ are constant values in the domain of A_i, and *op* is a comparison operator in $\{=, \neq, >, <, \geq, \leq\}$. For the sake of readability, in the following, we will use notation *Attr*(*cond*) to denote the set of attributes in the basic condition *cond*. To partially delegate the computation of the query to the providers storing F_1^e and F_2^e, q is translated into a set $\{q_1, q_2, q_u\}$ of queries operating at the provider storing F_1^e, at the provider storing F_2^e, and at the user side, respectively. This translatwion is based on the observation that the evaluation of basic conditions involving only attributes plaintext represented in F_1^e (F_2^e, respectively) can be delegated to the provider storing F_1^e (F_2^e, respectively). Conditions operating on encrypted attributes or on two attributes, say A_1 and A_2, with $A_1 \in F_1$ and $A_2 \in F_2$, must be evaluated by the user. Given query q, condition *Cond* in the WHERE clause is then split into three sub-conditions as follows:

- $Cond_1 = \bigwedge_i cond_i$: $Attr(cond_i) \subseteq F_1$ is the conjunction of basic conditions that involve only attributes in fragment F_1;
- $Cond_2 = \bigwedge_i cond_i$: $Attr(cond_i) \subseteq F_2$ is the conjunction of basic conditions that involve only attributes in fragment F_2;

– $Cond_u$=$\bigwedge_i cond_i$: $Attr(cond_i) \not\subseteq F_1$ and $Attr(cond_i) \not\subseteq F_2$ is the conjunction of basic conditions that either involve encrypted attributes or are of the form $(A_i\ op\ A_j)$, where $A_i \in F_1$ and $A_j \in F_2$ (or viceversa).

For instance, consider relation PATIENTS in Figure 1(a), its fragmentation in Figure 6 and query q="SELECT `Name` FROM `Patients` WHERE `Job`='lawyer' AND `Disease`='flu' AND `Ins`=100." $Cond_1$ includes basic condition `Job`='lawyer'; $Cond_2$ includes basic condition `Disease`='flu'; and $Cond_u$ includes basic condition `Ins`=100.

The evaluation of a query q on R can follow different strategies, depending on whether $Cond_1$ and $Cond_2$ are evaluated in parallel (Figure 8(a)) or in sequence (Figure 8(b)), as illustrated in the following.

– *Parallel strategy*. The two providers evaluate in parallel queries q_1 and q_2, which are in charge of returning the tuples in F_1^e and F_2^e satisfying conditions $Cond_1$ and $Cond_2$, respectively. Query q_1 (q_2, respectively) returns the tuple identifier `tid`, which is necessary to join its result R_1 (R_2, respectively) with the result of q_2 (q_1, respectively), and all those attributes included in F_1 (F_2, respectively) and in E that appear either in the SELECT clause of q, or in $Cond_u$. When the user receives both R_1 and R_2, she executes query q_u that computes the join between them, decrypts encrypted attributes, evaluates $Cond_u$, and projects the attributes in the SELECT clause of q. Figure 8(a) illustrates the translation of query q="SELECT `Name` FROM `Patients` WHERE `Job`='lawyer' AND `Disease`='flu' AND `Ins`=100" formulated over relation PATIENTS in Figure 1(a) into an equivalent set of queries operating on the fragments in Figure 6.
– *Sequential strategy*. With this strategy, one among the two queries q_1 and q_2 is sent to the corresponding provider first. Let us assume that the provider storing F_1^e goes first (the case where the provider managing F_2^e goes first is symmetric) and executes query q_1, which evaluates condition $Cond_1$ retuning attribute `tid` and all those attributes included in F_1 and in E that appear either in the SELECT clause of q, or in $Cond_u$. Upon receiving R_1, the user sends to the provider storing F_2^e the identifiers of the tuples in R_1. The provider then executes q_2, which evaluates $Cond_2$ on the tuples in F_2^e whose identifier is among the ones received from the user. The user finally computes the join between R_1 and R_2, evaluates $Cond_u$ and projects the attributes of interest. Figure 8(b) illustrates the translation of query q="SELECT `Name` FROM `Patients` WHERE `Job`='lawyer' AND `Disease`='flu' AND `Ins`=100" formulated over relation PATIENTS in Figure 1(a) into an equivalent set of queries operating on the fragments in Figure 6. (Values {4,6} in the WHERE clause of q_2 are the identifiers of the tuples satisfying $Cond_1$.)

The choice between the parallel and the sequential strategies depends on the performance they guarantee and on the resource that the user considers more valuable. In fact, the parallel strategy has the advantage of reducing the response time, while causing a higher communication cost than the sequential

Original query	Translated queries
$q :=$ SELECT Att FROM R WHERE $Cond$	$q_1 :=$ SELECT tid, $(Att \cup Attr(Cond_u)) \cap (F_1 \cup E)$ FROM F_1^e WHERE $Cond_1$ $q_2 :=$ SELECT tid, $(Att \cup Attr(Cond_u)) \cap (F_2 \cup E)$ FROM F_2^e WHERE $Cond_2$ $q_u :=$ SELECT Att FROM R_1 JOIN R_2 ON $R_1.tid = R_2.tid$ WHERE $Cond_u$
$q :=$ SELECT Name FROM Patients WHERE Job='lawyer' AND Disease='flu' AND Ins=100	$q_1 :=$ SELECT tid, Name, Ins^1 FROM F_1^e WHERE Job='lawyer' $q_2 :=$ SELECT tid, Ins^2 FROM F_2^e WHERE Disease='flu' $q_u :=$ SELECT Name FROM R_1 JOIN R_2 ON R_1.tid=R_2.tid WHERE $Decrypt(\text{Ins}^1, \text{Ins}^2)$=100

(a) Parallel strategy

Original query	Translated queries
$q :=$ SELECT Att FROM R WHERE $Cond$	$q_1 :=$ SELECT tid, $(Att \cup Attr(Cond_u)) \cap (F_1 \cup E)$ FROM F_1^e WHERE $Cond_1$ $q_2 :=$ SELECT tid, $(Att \cup Attr(Cond_u)) \cap (F_2 \cup E)$ FROM F_2^e WHERE (tid IN $R_1.tid$) AND $Cond_2$ $q_u :=$ SELECT Att FROM R_1 JOIN R_2 ON $R_1.tid = R_2.tid$ WHERE $Cond_u$
$q :=$ SELECT Name FROM Patients WHERE Job='lawyer' AND Disease='flu' AND Ins=100	$q_1 :=$ SELECT tid, Name, Ins^1 FROM F_1^e WHERE Job='lawyer' $q_2 :=$ SELECT tid, Ins^2 FROM F_2^e WHERE (tid IN {4,6}) AND Disease='FLU' $q_u :=$ SELECT Name FROM R_1 JOIN R_2 ON R_1.tid=R_2.tid WHERE $Decrypt(\text{Ins}^1, \text{Ins}^2)$=100

(b) Sequential strategy

Fig. 8. An example of query translation in the two can keep a secret scenario

strategy (R_2 is likely to be composed of a higher number of tuples). The choice of the provider that goes first in the sequential strategy instead depends only on

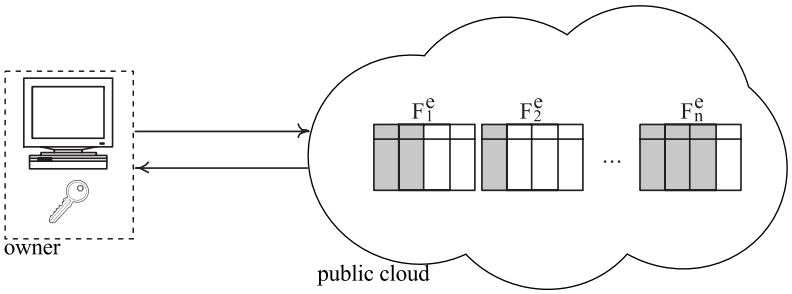

Fig. 9. Multiple fragments

the selectivity of $Cond_1$ and $Cond_2$, since it is preferable to evaluate the most selective condition first.

5 Multiple fragments

We present a solution based on encryption and fragmentation where data can be splitted among an arbitrary number of fragments [7,10], which may be possibly stored at the same provider (Figure 9).

Fragmentation model The goal of the proposal in [7] is to remove the limiting assumption of absence of collusion between the two providers characterizing the solution in [1]. The idea is therefore to compute a fragmentation (with no limits on the number of fragments composing it) in such a way that its fragments are not linkable, meaning that it is not possible for parties different from the data owner and authorized users to reconstruct the original relation and then also the sensitive values and associations. Being non linkable, fragments can be stored at a different providers, but also at the same provider, with no confidentiality risk.

The approach in [7] couples vertical fragmentation with encryption to satisfy confidentiality constraints. In particular, each singleton constraint $c=\{A\}$ is satisfied by encrypting the involved attribute A. Each association constraint $c=\{A_1, \ldots, A_n\}$ can instead be satisfied by either encrypting at least one among the $A_1, \ldots, A_n$ attributes, or by storing these attributes in different fragments. To prevent indirect violation of confidentiality constraints by joining fragments, fragments must be disjoint (i.e., no attribute can appear in more than one fragment). More formally, a *correct* fragmentation is defined as follows.

Definition 3 (Correct Fragmentation). *Let $R(A_1, \ldots, A_n)$ be a relation schema and $\mathcal{C}$ be a set of confidentiality constraints over it. A fragmentation $\mathcal{F} = \{F_1, \ldots, F_m\}$ is* correct *iff:*

- $\forall c \in \mathcal{C}$, $\forall F \in \mathcal{F}$: $c \not\subseteq F$ (*confidentiality*);
- $\forall F_i, F_j \in \mathcal{F}$, $i \neq j$: $F_i \cap F_j = \emptyset$ (*unlinkability*).

F_1^e

salt	enc	Name	Job
s_1^1	xTb:	Alice	teacher
s_1^2	o;!G	Bob	farmer
s_1^3	Ap'L	Carol	nurse
s_1^4	.u7t	David	lawyer
s_1^5	y"e3	Eric	secretary
s_1^6	(l1!	Fred	lawyer

F_2^e

salt	enc	Disease	Treatment
s_2^1	hg5=	flu	paracetamol
s_2^2	mB71	asthma	bronchodilators
s_2^3	:k?2	gastritis	antacids
s_2^4	Ql4,	angina	nitroglycerin
s_2^5	-kGd	flu	aspirin
s_2^6	p[Mz	diabetes	insulin

F_3^e

salt	enc	Race	Ins
s_3^1	bP5	white	160
s_3^2	*Cx	white	100
s_3^3	1Bny	asian	100
s_3^4	Oj)6	black	200
s_3^5	vT7/	black	100
s_3^6	l1fY	asian	180

Fig. 10. An example of correct fragmentation of relation PATIENTS in Figure 1(a) in the multiple fragments scenario

The first condition states that a fragment in $\mathcal{F}$ cannot contain all the attributes composing a confidentiality constraint. The second condition states that fragments must be disjoint. This approach has two advantages: *i)* being disjoint, all fragments $F_1, \ldots, F_n$ composing a fragmentation $\mathcal{F}$ are not linkable and can therefore be stored at the same provider; and *ii)* not imposing any limit on the number of fragments, association constraints can always be satisfied without encryption, thus increasing the *visibility* over data, with clear advantages for query evaluation. In fact, the plaintext representation of an attribute A in a fragment F permits the evaluation of conditions over A at the cloud provider storing F. For this reason, the approach in [7] aims at computing fragmentations that maximize visibility. A fragmentation maximizes visibility if each attribute A in R not appearing in a singleton constraint is plaintext represented in *at least* one fragment. Note however that, to satisfy the unlinkability condition, each attribute not appearing in a singleton constraint can belong to *at most* one fragment in a correct fragmentation. For instance, $\mathcal{F}$={{`Name`,`Job`}, {`Disease`,`Treatment`}, {`Race`,`Ins`}} is a correct fragmentation of relation PATIENTS in Figure 1(a) with respect to the constraints in Figure 1(b). This fragmentation maximizes visibility as all the attributes but `SSN`, which is sensitive per se (c_1), are plaintext represented in exactly one fragment.

At a physical level, each fragment $F_i = \{A_{i_1}, \ldots, A_{i_n}\}$ of a fragmentation $\mathcal{F}$ is represented by a *physical fragment* $F_i^e(\underline{salt}, enc, A_{i_1}, \ldots, A_{i_n})$, where:

- *salt* is the primary key of the relation and contains a randomly chosen value;
- *enc* is an attribute storing the encrypted attributes in $R \backslash \{A_{i_1}, \ldots, A_{i_n}\}$;
- $\{A_{i_1}, \ldots, A_{i_n}\}$ is the set of attributes composing fragment F_i.

Each tuple t in r is represented by a tuple in each of the physical fragments $\{F_1^e, \ldots, F_m^e\}$ corresponding to the fragments $\{F_1, \ldots, F_m\}$ in $\mathcal{F}$. Tuple t^e representing t in F_i^e is such that: $t^e[salt]$ is a random value; $t^e[enc]$ is computed as $Enc(t[R \setminus F_i] \oplus t[salt], k)$, with $\oplus$ the binary XOR operator; and $t^e[A]$=$t[A]$, $\forall A \in F$. Note that the attributes not appearing in plaintext in F^e are combined with a random salt before encryption to prevent frequency attacks [33]. Since each physical fragment stores (either plaintext or encrypted) all the attributes in R, every query can be evaluated on a single fragment. Figure 10 illustrates the physical fragments representing fragmentation $\mathcal{F}$={{`Name`,`Job`}, {`Disease`,`Treatment`}, {`Race`,`Ins`}} of relation PATIENTS in Figure 1(a).

Metric	Quality function
Number of fragments	$card(\mathcal{F})$
Affinity	$\sum_{k=1}^{n} aff(F_k)$ where $aff(F_k) = \sum_{A_i,A_j \in F_k, i<j} M[A_i, A_j]$, $k = 1, \ldots, n$
Query evaluation cost	$\sum_{i=1}^{m} freq(q_i) \cdot cost(q_i,\mathcal{F})$ where $cost(q_i,\mathcal{F}) = Min(cost(q_i,F_j),\ j = 1, \ldots, n)$ and $cost(q_i,F_j) = S(q_i, F_j) \cdot \lvert r \rvert \cdot\ size(t_j)$, $i = 1, \ldots, m$ and $j = 1, \ldots, n$

Fig. 11. Classification of the metrics in the multiple fragments scenario

Fragmentation metrics Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, there may be different correct fragmentations that maximize visibility. As an example, a fragmentation $\mathcal{F}$ where each attribute in R that does not appear in a singleton constraint is stored in a different fragment is correct and maximizes visibility. However, this solution causes an excessive fragmentation making the evaluation of queries involving more than one attribute expensive. We now present different metrics (see Figure 11) that can be used to assess the quality of a fragmentation in terms of the query evaluation overhead caused to users.

- *Minimal fragmentation* (e.g., [7]). A simple metric for evaluating the quality of a fragmentation consists in minimizing the number of fragments thus avoiding excessive fragmentation. Intuitively, a fragmentation with a lower number of fragments is likely to store more attributes in the same fragment, clearly favoring the evaluation of queries that involve these attributes (also together). For instance, both $\mathcal{F}$={{`Name`,`Job`}, {`Disease`,`Treatment`}, {`Race`,`Ins`}} and $\mathcal{F}'$={{`Name`,`Job`}, {`Disease`},{`Treatment`}, {`Race`,`Ins`}} are correct fragmentations of relation PATIENTS in Figure 1(a). However, $\mathcal{F}$ is preferable to $\mathcal{F}'$ because it efficiently supports the evaluation of queries involving, both `Disease` and `Treatment`. Two different notions of minimality have been proposed: *minimality* (i.e., composed of the minimum number of fragments [3,12]) and *local minimality* (i.e., composed of fragments that cannot be merged without violating constraints [7,12]). We note that, while a locally minimal fragmentation might not be composed of the minimum number of fragments, a minimal fragmentation is indeed also locally minimal (i.e., merging any of its fragments violates at least a constraint).
- *Maximum affinity* (e.g., [10]). A more precise assessment of the quality of a fragmentation is based on the *affinity* between attributes. The affinity between two attributes quantifies the performance advantage in query evaluation that can be obtained by storing them in the same fragment [31]. Attributes with high affinity are expected to be frequently involved together in queries. Therefore, the higher the affinity, the higher the advantage in query evaluation of having the attributes stored in the same fragment. Attribute

	Name	Race	Job	Disease	Treatment	Ins
Name		10	30	10	10	10
Race			10	10	10	40
Job				10	10	10
Disease					25	10
Treatment						10
Ins						

Fig. 12. An example of affinity matrix

affinity can be modeled by an *affinity matrix* M, which is a symmetric matrix with a row and a column for each attribute that do not appear in a singleton constraint. Each cell $M[A_i,A_j]$, with $i \neq j$, represents the benefit obtained by storing attributes A_i and A_j in the same fragment. For instance, Figure 12 illustrates an example of affinity matrix for relation PATIENTS in Figure 1(a). Fragmentations that keep in the same fragment attributes with high affinity are to be preferred over fragmentations that split them in different fragments. The quality of a fragmentation $\mathcal{F}$ is measured as the sum of the affinity of the fragments composing it, where the affinity of a fragment F is obtained by summing the affinities of the pairs of attributes in F. As an example, consider relation PATIENTS in Figure 1(a), the fragmentation in Figure 10, and the affinity matrix in Figure 12. The quality of $\mathcal{F}$ is computed as: $M[\texttt{Name}, \texttt{Job}] + M[\texttt{Disease}, \texttt{Treatment}] + M[\texttt{Race}, \texttt{Ins}] = 30 + 25 + 40 = 95$.

– *Minimum query evaluation cost* (e.g., [8]). Another possible metric is based on the definition of a query cost model, which can be used to evaluate the cost of executing a representative set of queries over fragments. This metric, compared with the affinity metric, has the advantage of taking into consideration also the benefit of storing in the same fragment arbitrary sets of attributes (instead of pairs thereof). The adoption of this metric requires the availability of the query workload $\mathcal{Q}$ of the system, which is a set $\{q_1, \ldots, q_m\}$ of queries along with their execution frequency $freq(q_i)$, $i = 1, \ldots, m$. The proposal in [8] assumes that queries in $\mathcal{Q}$ are of the form "SELECT $A_{i_1}, \ldots, A_{i_n}$ FROM R WHERE $\bigwedge_{j=1}^{n}$ (A_j IN V_j)" with V_j a set of values in the domain of attribute A_j. The quality of a fragmentation $\mathcal{F}$ then depends on the cost of executing the queries in $\mathcal{Q}$, properly weighted by their frequency, over the fragments in $\mathcal{F}$. Since each physical fragment stores, either plaintext or encrypted, all the attributes in R, the cost of evaluating a query q over $\mathcal{F}$ is the minimum among the costs of evaluating q over each physical fragment F^e in $\mathcal{F}$. The cost of evaluating q on F^e is estimated by the size of the result returned to the user, because the costs of communication, decryption, and evaluation of conditions on encrypted attributes at the user side are more expensive than the computational costs at the provider side. Hence, the cost $cost(q_i,F_j)$ of executing q_i on F_j is computed as $S(q_i, F_j) \cdot |r| \cdot size(t_j)$, where $S(q_i, F_j)$ is the selectivity of query q_i, $|r|$ is the number of tuples in r, and $size(t_j)$ is the size of the attributes appearing in the SELECT clause of q_i and the size of attribute *enc* if there is the need of accessing attributes not appearing in plaintext in F_j. The selectivity of condition A

IN $V = \{v_1, \ldots, v_n\}$ is an estimate of the ratio of the number of tuples in F that satisfy the condition over the total number of tuples in r. If attribute A does not appear in plaintext in F, the selectivity is set to 1. Consider, as an example, a query workload composed of two queries: q_1="SELECT * FROM `Patients` WHERE `Job`='teacher' AND `Race`='asian'", with frequency 30; and q_2="SELECT* FROM `Patients` WHERE `Job`='lawyer' AND `Disease`='flu'", with frequency 70. The fragmentation in Figure 10 implies a query evaluation cost $cost(\mathcal{Q}, \mathcal{F}) = cost(q_1, \mathcal{F}) \cdot freq(q_1) + cost(q_2, \mathcal{F}) \cdot freq(q_2) = 1/6 \cdot 6 \cdot 1 \cdot 30 + 1/3 \cdot 6 \cdot 1 \cdot 70 = 170$. In fact, assuming that the size of the tuples is the same for all the fragments and is equal to 1, the fragment that minimizes query evaluation cost for q_1 is F_1, and are both F_1 and F_2 for q_2. Indeed, the most selective condition in q_1 operates on attribute `Job`, while the two conditions in q_2 are equally selective.

Computing an optimal fragmentation Regardless of the metric adopted to evaluate the quality of a fragmentation, the problem of computing an optimal fragmentation is NP-hard (the minimum hypergraph coloring problem reduces to it in polynomial time [10]). Hence, the time complexity of any algorithm able to compute an optimal fragmentation is exponential in the number of attributes in R. In the following, we briefly survey exact and heuristic algorithms proposed for efficiently computing a correct and optimal (according to a chosen metric) fragmentation.

- *Minimal fragmentation* (e.g., [3,7,12]). Both exact and heuristic algorithms have been proposed to the aim of avoiding an excessive fragmentation and producing minimal or locally minimal fragmentations. The exact algorithms in [3,12], proposed to produce a minimal fragmentation, rely on a logical modeling of the problem. The attributes in R are interpreted as Boolean variables, and each confidentiality constraint c=$\{A_1,\ldots,A_n\}$ in $\mathcal{C}$ as a Boolean formula representing the conjunction $A_1 \wedge \ldots \wedge A_n$ of the attributes composing it. A fragment F of R is a truth assignment that assigns *true* to the variables representing the attributes in the fragment, and *false* to the other variables. A fragmentation $\mathcal{F}$ is a set of truth assignments that satisfy all the constraints in $\mathcal{C}$, and such that each variable A is assigned *true* in at most one fragment F in $\mathcal{F}$. Two approaches have been studied to compute a set of truth assignments representing a correct fragmentation that use a SAT (SATisfiability) and an OBDD (Ordered Binary Decision Diagram) formulation of the fragmentation problem. The adoption of SAT solvers has been proposed in [3] to compute a fragmentation composed of the minimum the number of fragments. To this aim, a SAT solver able to compute a correct fragmentation composed of n fragments is iteratively invoked. At the first iteration, n is set to 1. It is then incremented by 1 at each iteration. The iteration stops when the SAT solver finds a correct fragmentation. The adoption of the OBDD data structure to represent confidentiality constraints and efficiently compute fragments (i.e., truth assignments) satisfying constraints

has been proposed in [12]. The problem of computing a fragmentation composed of the minimum number of fragments is translated into the problem of computing a maximum weighted clique over a *fragmentation graph*. The fragmentation graph models fragments, efficiently computed using OBDDs, that satisfy all the confidentiality constraints and a subset of the visibility constraints (i.e., required views over the data) defined in the system. Another heuristic approach for computing a locally minimal fragmentation has been proposed in [7]. The algorithm starts from an empty fragmentation $\mathcal{F}$ and tries to insert each attribute A in R (non involved in a singleton constraint) into a fragment $F \in \mathcal{F}$. If A cannot be inserted into any fragment in $\mathcal{F}$ without violating constraints, a new fragment $F'=\{A\}$ is created and inserted into $\mathcal{F}$. The attributes are considered in decreasing order of the number of constraints in which they are involved (i.e., attributes appearing in a higher number of constraints are considered first).

- *Maximum affinity* (e.g., [10]). The greedy approach proposed in [10] takes advantage of the affinity matrix M previously illustrated in this section to compute a fragmentation that maximizes affinity. The proposed technique starts with a fragmentation $\mathcal{F}$ where each attribute A that does not appear in a singleton constraint belongs to a different fragment $F \in \mathcal{F}$. At each iteration, the algorithm merges the pair of fragments $\langle F_i, F_j \rangle$ with highest affinity according to M, provided no constraint is violated. The algorithm terminates when no further merge is possible.
- *Minimum query evaluation cost* (e.g., [8]). The exact algorithm proposed in [8] to minimize the cost of query execution is based on an efficient visit of the solution space of the fragmentation problem, which is represented through a lattice $(S_F, \preceq)$. Set S_F includes all the fragmentations of relation R composed of disjoint fragments; $\preceq$ is a dominance relationship between fragmentations where $\mathcal{F} \preceq \mathcal{F}'$ iff $\mathcal{F}$ can be obtained by merging fragments in $\mathcal{F}'$. The visit of the fragmentation lattice is based on two nice properties of the dominance relationship: *i)* given a non-correct fragmentation $\mathcal{F}'$, any fragmentation $\mathcal{F}$ such that $\mathcal{F} \preceq \mathcal{F}'$ is not correct; and *ii)* given two fragmentations $\mathcal{F}$ and $\mathcal{F}'$ such that $\mathcal{F} \preceq \mathcal{F}'$, the query evaluation cost of $\mathcal{F}'$ is higher than the cost of $\mathcal{F}$ (i.e., the cost is monotonic with the dominance relationship). A heuristic algorithm exploiting the fragmentation lattice has also been proposed [8].

Query evaluation Since each physical fragment stores (either plaintext of encrypted) all the attributes in R, a query q can be evaluated on any physical fragment. However, the performance clearly depends on the fragment that is chosen to evaluate q. Given the set Att of attributes involved in q, it is intuitively more convenient to evaluate q over a physical fragment F_i^e that stores attributes in Att (or a subset thereof) in the clear, rather than over a fragment F_j^e where attributes in Att are encrypted. In fact, choosing F_j^e would require the user to download the whole fragment from the cloud and to locally evaluate the query

Original query	Translated queries
$q :=$ SELECT Att FROM R WHERE $Cond$	$q_p :=$ SELECT $salt$, enc, $Att \cap F$ FROM F^e WHERE $Cond_p$ $q_u :=$ SELECT Att FROM $Decrypt(R_p.enc \oplus salt, k)$ WHERE $Cond_u$
$q :=$ SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Job`='teacher'	$q_p :=$ SELECT `salt`, `enc`, `Name` FROM F_1^e WHERE `Job`='teacher' $q_u :=$ SELECT `Name` FROM $Decrypt(R_p.\texttt{enc} \oplus \texttt{salt}, k)$ WHERE `Disease`='flu'

Fig. 13. An example of query translation in the multiple fragments scenario

(after the encrypted attributes have been decrypted). Instead, resorting to F_i^e permits to (partially) delegate to the provider the query evaluation task.

The solution in [7] considers queries q of the form SELECT Att FROM R WHERE $Cond$, with Att a set of attributes in R and $Cond = \bigwedge_i cond_i$ a conjunction of basic conditions of the form $(A_i\ op\ v)$, $(A_i\ op\ A_j)$, or $(A_i$ IN $\{v_i, \ldots, v_k\})$, where $A_i, A_j \in R$, $\{v, v_1, \ldots, v_k\}$ are constant values in the domain of A_i, and op is a comparison operator in $\{=, \neq, >, <, \geq, \leq\}$. Let us assume that query q is evaluated over physical fragment F^e. The basic conditions in the WHERE clause of q operating on attributes in F (i.e., attributes appearing in plaintext in F^e) can be evaluated by the provider while the basic conditions operating on attributes not included in F (i.e., encrypted attributes in F^e) must be evaluated by the user who knows the encryption key. To translate q into an equivalent set of queries operating on F^e, $Cond$ is split in sub-conditions $Cond_p$ and $Cond_u$, as follows:

- $Cond_p = \bigwedge_i cond_i : Attr(cond_i) \subseteq F$ is the conjunction of the basic conditions involving only attributes plaintext stored in the chosen fragment;
- $Cond_u = \bigwedge_i cond_i : Attr(cond_i) \not\subseteq F$ is the conjunction of the basic conditions that involve at least one attribute that appears encrypted in the chosen fragment.

For instance, consider the fragmentation in Figure 10, and assume that query "SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Job`='teacher'" is evaluated over fragment F_1^e. In this case, $Cond$ includes conditions `Disease`='flu' and `Job`='teacher'. Since attribute `Job` belongs to F_1 while attribute `Disease` does not, $Cond_p$ includes condition `Job`='teacher', and $Cond_u$ includes condition `Disease`='flu'.

After conditions in $Cond$ have been classified between $Cond_p$ and $Cond_u$, query q is translated in two queries, as illustrated in Figure 13. Query q_p, ex-

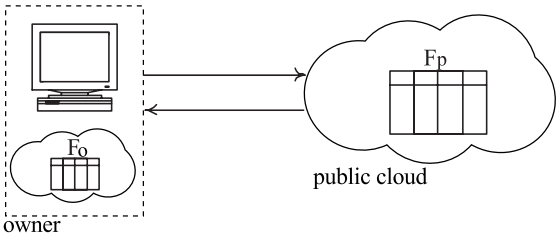

Fig. 14. Keep a few

ecuted at the provider side, operates on the selected physical fragment F^e and evaluates condition $Cond_p$. When the user receives the result R_p of query q_p, it decrypts attribute *enc* and evaluates, on the resulting tuples, query q_u. Query q_u evaluates condition $Cond_u$ and projects the attributes in *Att*. Note that if $Cond_u$ is empty and all the attributes in the SELECT clause of q belong to F, then q_u does not need to be executed and q_p does not need to return attributes *salt* and *enc* (since the result R_p returned by q_p already coincides with the result of the original query q). Consider, as an example, relation PATIENTS in Figure 1(a), the fragmentation in Figure 10, and query q="SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Job`='teacher'", returning the names of the teachers suffering from flu. While, in principle, the query might be evaluated using any of the three fragments, F_1^e and F_3^e are more convenient than F_2^e because F_2^e does not include any of the attributes involved in the conditions of q. Figure 13 illustrates the translation of q in the corresponding sub-queries operating at the provider side (i.e., q_p) and at the user side (i.e., q_u) using F_1^e.

6 Keep a few

We present a solution completely departing from encryption where a trusted party (the owner) is involved in storing a portion of the data (Figure 14). Specifically, data are splitted into two fragments, one stored at the data owner side, and one stored at an external provider so that the fragment managed by the provider does not violate the confidentiality constraints [4,9,11].

Fragmentation model Sensitive associations are protected by the approaches discussed in previous sections by encrypting (a portion of) the original relation and/or by splitting its content into non-linkable fragments. The approach in [9] departs from encryption, and protects sensitive associations relying on owner-side storage to satisfy confidentiality constraints. According to this proposal, relation R is fragmented generating a pair $\mathcal{F} = \langle F_o, F_p \rangle$ of fragments, with F_o stored at the data owner and F_p stored at a cloud provider. This solution satisfies singleton constraints c=$\{A\}$ by storing A at the owner. Similarly, it satisfies association constraints c=$\{A_1, \ldots, A_n\}$ by storing at least one among $\{A_1, \ldots, A_n\}$ at the owner. Formally, a *correct fragmentation* is defined as follows.

F_o^e

tid	SSN	Name	Ins
1	123-45-6789	Alice	160
2	234-56-7890	Bob	100
3	345-67-8901	Carol	100
4	456-7 8-9012	David	200
5	567-89-0123	Eric	100
6	678-90-1234	Fred	180

F_s^e

tid	Race	Job	Disease	Treatment
1	white	teacher	flu	paracetamol
2	while	farmer	asthma	bronchodilators
3	asian	nurse	gastritis	antacids
4	black	lawyer	angina	nitroglycerin
5	black	secretary	flu	aspirin
6	asian	lawyer	diabetes	insulin

Fig. 15. An example of a correct fragmentation of relation PATIENTS in Figure 1(a) in the keep a few scenario

Definition 4 (Correct Fragmentation). *Let $R(A_1, \ldots, A_n)$ be a relation schema and $\mathcal{C}$ be a set of confidentiality constraints over it. A fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ is* correct *iff:*

- $\forall c \in \mathcal{C}, c \not\subseteq F_p$ (*confidentiality*);
- $F_o \cup F_p = R$ (*losslessness*).

The first condition states that fragment F_p cannot contain all the attributes composing a confidentiality constraint. This condition must hold only for F_p, since F_o is stored at the data owner and is therefore accessible to authorized users only. The second condition demands that all attributes in R are represented at the data owner or at the cloud provider, thus guaranteeing losslessness of the fragmentation. Although, in principle, F_o might include attributes appearing in F_p, this redundancy is not necessary and might be expensive for the data owner (both in terms of storage and computation). Fragments are then required to be disjoint (i.e., $F_o \cap F_p = \emptyset$). For instance, $\mathcal{F} = \langle F_o, F_p \rangle$ with F_o={`SSN`,`Name`,`Ins`} and F_p={`Race`,`Job`,`Disease`,`Treatment`} represents a correct fragmentation of relation PATIENTS in Figure 1(a) with respect to the confidentiality constraints in Figure 1(b).

At the physical level, fragments F_o and F_p must have a common key attribute to permit authorized users to correctly reconstruct the content of relation r. This attribute can be either the primary key of relation R, if it is not sensitive, or an attribute that does not belong to the schema of R and that is added to both F_o and F_p to this purpose. Assuming that the primary key of R cannot be publicly released, a fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$, with $F_o = \{A_{o_1}, \ldots, A_{o_i}\}$ and $F_p = \{A_{p_1}, \ldots, A_{p_j}\}$, is translated into physical fragments $F_o^e(\underline{tid}, A_{o_1}, \ldots, A_{o_i})$ and $F_p^e(\underline{tid}, A_{p_1}, \ldots, A_{p_j})$, where *tid* is a randomly generated tuple identifier. Figure 15 illustrates the physical fragments representing fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ with F_o={`SSN`,`Name`,`Ins`} and F_p={`Race`,`Job`,`Disease`,`Treatment`} of relation PATIENTS in Figure 1(a). Note that at least one attribute of each constraint in Figure 1(b) is in F_o.

Fragmentation metrics Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, there may exist different correct fragmentations that are non-redundant. For instance, consider a fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ where $F_o = R$ and $F_p = \emptyset$. This fragmentation is correct and non-redundant, but

Attribute A	$size(A)$
SSN	10
Name	20
Race	5
Job	18
Disease	18
Treatment	30
Ins	8

(a)

Query q	$freq(q)$	$Attr(q)$	$Cond_q$
q_1	20	Job, Disease	⟨Job⟩, ⟨Disease⟩
q_2	30	Disease, Treatment	⟨Disease⟩, ⟨Treatment⟩
q_3	40	Job, Ins	⟨Job⟩, ⟨Ins⟩
q_4	10	SSN, Ins, Disease	⟨SSN⟩, ⟨Ins⟩, ⟨Disease⟩

(b)

Fig. 16. An example of size of attributes (a) and query workload (b) for relation PATIENTS in Figure 1(a)

it does not take advantage of outsourcing as no storage and/or computation is delegated to the cloud provider. To maximize the advantages for the data owner, she must push to the cloud provider as much as possible of the storage and computation workload necessary for the management of her data. To this aim, it is necessary to properly measure the storage, computation, and communication overhead caused to the data owner by the storage and management of fragment F_o. In the following, we illustrate the metrics that can be adopted to assess the quality of a fragmentation, depending on the resource that the data owner values more and on the information available about the system workload at initialization time [9].

- *Minimal fragmentation*. The most straightforward metric consists in counting the number of attributes in F_o. Intuitively, a fragment composed of a lower number of attributes is likely to be smaller (reducing the storage occupation), and to be involved in a lower number of queries (reducing the computation and communication overhead).
- *Minimal size of attributes*. If the data owner aims at limiting the storage occupation at the provider side, the most effective metric to assess the quality of a fragmentation measures the size of F_o. The storage occupation of F_o is computed as the sum of the size of the attributes composing it. For instance, suppose that the size of the attributes of relation PATIENTS in Figure 1(a) is as summarized in Figure 16(a). The size of fragment F_o in Figure 15 is $size(\texttt{SSN})+size(\texttt{Name})+size(\texttt{Ins}) = 10 + 20 + 8 = 38$.
- *Minimal number of queries*. The computation and communication overhead at the data owner side can be measured as the number of queries whose evaluation requires the owner's intervention (i.e., queries involving at least one attribute in F_o). The adoption of this metric requires the knowledge of the query workload $\mathcal{Q}$ characterizing the system that, in this scenario, is a set $\{q_1, \ldots, q_m\}$ of representative queries, along with their frequency $freq(q_i)$, $i = 1, \ldots, m$. For instance, the first three columns in Figure 16(b) represent a query workload for relation PATIENTS in Figure 1(a). The cost of a fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ is computed as the sum of the frequencies of the queries including at least one attribute in F_o. With respect to the workload in Figure 16(b), the fragmentation in Figure 15 requires the evaluation of

	Metric	Quality function
Storage	Number of attributes	$card(F_o)$
	Size of attributes	$\sum_{A \in F_o} size(A)$
Computation and communication	Number of queries	$\sum_{q \in \mathcal{Q}} freq(q)$ *s.t.* $Attr(q) \cap F_o \neq \emptyset$
	Number of conditions	$\sum_{cond \in Cond(\mathcal{Q})} freq(cond)$ *s.t.* $cond \cap F_o \neq \emptyset$

Fig. 17. Classification of the metrics in the keep a few scenario

$freq(q_3)$+$freq(q_4)$ = 50 queries at the data owner, since q_3 and q_4 involve attributes in F_o.

- *Minimal number of conditions.* A more precise metric measuring the computation overhead of the data owner considers, instead of the number of queries, the number of conditions she should evaluate. In fact, the presence of multiple conditions in the same query operating on F_o causes a higher computation overhead for the data owner. To adopt this metric, it is necessary to know, besides the frequency $freq(q_i)$ of each query q_i in the query workload $\mathcal{Q}$, also the conditions, denoted $Cond(q_i)$, composing it. The quality of $\mathcal{F} = \langle F_o, F_p \rangle$ is then computed as the sum of the frequencies of the conditions in $\mathcal{Q}$ involving attributes in F_o. For instance, the first, second, and fourth column of Figure 16(c) represent a possible workload profile for relation PATIENTS in Figure 1(a). With respect to this workload, the fragmentation in Figure 15 requires the evaluation of $freq(\langle \texttt{SSN} \rangle)$+$freq(\langle \texttt{Ins} \rangle)$ $= 10 + (40 + 10) = 60$ conditions at the data owner side.

Figure 17 summarizes the formal definition of the metrics illustrated above. Note that the adoption of each metric is subject to the knowledge of different information about relation r and the query workload expected for the system.

Computing an optimal fragmentation The problem of computing an optimal fragmentation that minimizes the storage or the computation and communication costs for the data owner is NP-hard (the minimum hitting set problem reduces to it in polynomial time [9]). The heuristic approach proposed to compute a good fragmentation is based on the nice property that all the metrics illustrated above are monotonic in the number of attributes in F_o (i.e., the cost of a fragmentation $\mathcal{F}$ increases when an attribute is moved from F_p to F_o). Hence, the same heuristics applies to all the four metrics in Figure 17. The algorithm proposed in [9] aims at computing a *locally minimal fragmentation* $\mathcal{F} = \langle F_o, F_p \rangle$, which is defined as a fragmentation where no attribute can be moved from F_o to F_p without violating confidentiality constraints. The algorithm first inserts into F_o all the attributes that are considered sensitive per se (i.e., the attributes involved in singleton constraints). The remaining attributes, which initially belong to F_p, are organized in a priority queue. The priority of an

attribute A depends on: *i)* the number of constraints that would be solved moving A to F_o, and *ii)* the cost that the data owner would pay to move A to F_o. The algorithm iteratively extracts from the queue the attribute A with highest priority (i.e., the attribute with minimum cost per solved constraint) and inserts it into F_o. The iteration stops when either all the constraints are satisfied or the queue is empty (i.e., F_o=R and F_p=$\emptyset$). The algorithm finally tries to move each attribute in F_o to F_p, to guarantee minimality of the computed fragmentation.

Query evaluation A query q formulated over the original relation r must be translated into an equivalent set of queries operating on $\mathcal{F} = \langle F_o, F_p \rangle$. The solution in [9] considers queries q of the form SELECT Att FROM R WHERE $Cond$, with Att a set of attributes in R and $Cond = \bigwedge_i cond_i$ a conjunction of basic conditions of the form $(A_i\ op\ v)$, $(A_i\ op\ A_j)$, or $(A_i$ IN $\{v_i, \ldots, v_k\})$, where $A, A_i, A_j \in R$, $\{v, v_1, \ldots, v_k\}$ are constant values in the domain of A_i, and op is a comparison operator in $\{=, \neq, >, <, \geq, \leq\}$. Although in principle the data owner can evaluate, any query q formulated by the users, such a solution should be avoided when possible as it would reduce the advantages of resorting to a cloud provider for partial data storage and management. In fact, the cloud provider shluld be delegated for the evaluation of all those conditions operating on attributes in F_p. Given query q, the approach in [9] first splits $Cond$ in three sub-conditions, $Cond_o$, $Cond_p$, and $Cond_{po}$, depending on the attributes involved in each basic condition, as follows:

- $Cond_o = \bigwedge_i cond_i : Attr(cond_i) \subseteq F_o$ is the conjunction of basic conditions that involve only attributes stored at the data owner, which can be evaluated only by the owner;
- $Cond_p = \bigwedge_i cond_i : Attr(cond_i) \subseteq F_p$ is the conjunction of basic conditions that involve only attributes stored at the cloud provider, which can be evaluated by the provider;
- $Cond_{po} = \bigwedge_i cond_i : Attr(cond_i) \cap F_o \neq \emptyset$ and $Attr(cond_i) \cap F_p \neq \emptyset$ is the conjunction of basic conditions of the form $(A_i\ op\ A_j)$, where $A_i \in F_o$ and $A_j \in F_p$, which can be evaluated only by the data owner, with the support of the provider.

For instance, consider relation PATIENTS in Figure 1(a), its fragmentation in Figure 15 and query q="SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Ins`=100". $Cond_p$ includes condition `Disease`='flu'; $Cond_o$ includes condition `Ins`=100; and $Cond_{po}$ is empty.

The evaluation of a query q on R can follow the *provider-owner* or the *owner-provider* strategies, depending on the order in which $Cond_p$, $Cond_o$, and $Cond_{po}$ are evaluated (see Figure 18).

- *Provider-Owner strategy*. This strategy first evaluates condition $Cond_p$ at the provider side and then evaluates both $Cond_o$ and $Cond_{po}$ at the data owner side. Query q is translated into two equivalent queries q_p and q_o, as

Original query	Translated queries
q := SELECT Att FROM R WHERE $Cond$	q_p := SELECT tid, $(Att \cup Attr(Cond_{po})\}) \cap F_p$ FROM F_p^e WHERE $Cond_p$ q_o := SELECT Att FROM F_o^e JOIN R_p ON $F_o^e.tid{=}R_p.tid$ WHERE $Cond_o$ AND $Cond_{po}$
q := SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Ins`=100	q_p := SELECT `tid`, `Name` FROM F_p^e WHERE `Disease`='flu' q_o := SELECT `Name` FROM F_p^e JOIN R_p ON F_p^e.`tid`=R_p.`tid` WHERE `Ins`=100

(a) Provider-Owner strategy

Original query	Translated queries
q := SELECT Att FROM R WHERE $Cond$	q_o := SELECT tid FROM F_o^e WHERE $Cond_o$ q_p := SELECT $(Att \cup Attr(Cond_{po}))\cap F_p$ FROM F_p^e WHERE $(tid$ IN $R_o)$ AND $Cond_p$ q_{po} := SELECT Att FROM F_o^e JOIN R_p ON $F_o^e.tid{=}R_p.tid$ WHERE $Cond_{po}$
q := SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Ins`=100	q_o := SELECT `tid` FROM F_o^e WHERE `Ins`=100 q_p := SELECT `tid`, `Name` FROM F_p^e WHERE (`tid` IN {2,3,5}) AND `Disease`='flu' q_{po} := SELECT `Name` FROM F_o^e JOIN R_p ON F_o^e.`tid`=R_p.`tid`

(b) Owner-Provider strategy

Fig. 18. An example of query translation in the keep a few scenario

illustrated in Figure 18(a). Query q_p operates on F_p^e and evaluates condition $Cond_p$. It returns to the data owner the tuple identifier `tid` (which is necessary to join the result R_p of q_p and F_o^e) and the attributes in F_p^e that

appear in the SELECT clause of q or in $Cond_{po}$. When the data owner receives R_p, she executes query q_{po} that computes the join between R_p and F_o^e, evaluates $Cond_o$ and $Cond_{po}$, and projects the attributes in the SELECT clause of q. The data owner finally returns the result R_{po} of q_{po} to the user. As an example, Figures 18(a) illustrate the translation of query q="SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Ins`=100" formulated over relation PATIENTS in Figure 1(a) into an equivalent set of queries operating on the fragmentation in Figure 15.

– *Owner-Provider strategy*. This strategy first evaluates $Cond_o$ at the data owner, then evaluates condition $Cond_p$ at the provider side, and finally evaluates $Cond_{po}$ again at the data owner side. Query q is then translated into three queries, as illustrated in Figure 18(b). Query q_o operates on F_o^e, evaluates condition $Cond_o$, and projects attribute `tid` only. The result R_o of this query, computed by the data owner, is sent to the cloud provider that executes query q_p on the join between R_o and F_p^e. Query q_p evaluates condition $Cond_p$, and returns attribute `tid` and the attributes in F_p^e that appear in the SELECT clause of q or in $Cond_{po}$. The provider returns the result R_p of q_p to the data owner, who evaluates q_{po} on the join between R_p and F_o^e. Query q_{po} evaluates $Cond_{po}$ and projects the attributes in the SELECT clause of q. The data owner finally returns the result R_{po} of q_{po} to the user. As an example, Figures 18(b) illustrate the translation of query q="SELECT `Name` FROM `Patients` WHERE `Disease`='flu' AND `Ins`=100" formulated over relation PATIENTS in Figure 1(a) into an equivalent set of queries operating on the fragmentation in Figure 15 (values {2,3,5} in the WHERE clause of q_p represent the identifiers of the tuples satisfying $Cond_o$=`Ins`=100).

In the choice between these two strategies, it is necessary to take into consideration (besides performance) the risk of leakage of sensitive information that the Owner-Provider strategy may cause. In fact, if the provider knows the query q formulated by the user, this strategy reveals to the provider which are the tuples in F_p that satisfy $Cond_o$, even if the provider is not authorized to see the content of attributes in F_o.

7 Conclusions

Users as well as private and public organizations are more and more often relying on cloud providers to store and manage their data, enjoying economic advantages and high data availability. Although appealing, outsourcing the storage and management of data to the cloud introduces risks for data confidentiality, which could still represent a major obstacle to the wide adoption of cloud computing. In this chapter, we focused on the problem of protecting the confidentiality of data stored at external cloud providers. We first described the confidentiality requirements that may need to be considered and enforced before moving the data in the cloud. We then surveyed different approaches that have been proposed for enforcing such confidentiality requirements, using data encryption and

fragmentation, either by themselves or in combination. In addition to the data confidentiality issue treated in this chapter, other issues that might need to be addressed when relying on external cloud providers for data storage or computation include: data integrity, and availability; protection against external attacks; selective access to the data; fault tolerance management; the specification of security requirements on task/resource allocation in a cloud; query privacy; and query and computation integrity (e.g., [15,17,18,19,23,28,29,30]).

Acknowledgements This work was supported in part by the European Commission under the project "ABC4EU" (FP7-312797) and the Italian Ministry of Research within PRIN 2010-2011 project "GenData 2020" (2010RTFWBH).

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: Proc. of CIDR 2005. Asilomar, CA, USA (January 2005)
2. Agrawal, R., Kierman, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of SIGMOD 2004. Paris, France (June 2004)
3. Benedikt, M., Bourhis, P., Ley, C.: Querying schemas with access restrictions. Proc. of VLDB Endowment 5(7), 634–645 (March 2012)
4. Biskup, J., Preuß, M., Wiese, L.: On the inference-proofness of database fragmentation satisfying confidentiality constraints. In: Proc. of ISC 2011. Xi'an, China (October 2011)
5. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC 8(1), 119–152 (February 2005)
6. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Proc. of ACNS 2005. New York, NY, USA (June 2005)
7. 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: Proc. of ESORICS 2007. Dresden, Germany (September 2007)
8. 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: Proc. of ICDCS 2009. Montreal, Canada (June 2009)
9. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a few: Outsourcing data while maintaining confidentiality. In: Proc. of ESORICS 2009. Saint Malo, France (September 2009)
10. 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 TISSEC 13(3), 22:1–22:33 (July 2010)
11. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Selective data outsourcing for enforcing privacy. JCS 19(3), 531–566 (2011)
12. 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. JCS 20(5), 463–508 (2012)

13. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: Improved definitions and efficient constructions. In: Proc. of CCS 2006. Alexandria, VA, USA (October - November 2006)
14. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of CCS 2003. Washington, DC, USA (October 2003)
15. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Enforcing dynamic write privileges in data outsourcing. Computers & Security 39, 47–63 (November 2013)
16. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Fragmentation in presence of data dependencies. IEEE TDSC (2014), to appear
17. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM TODS 35(2), 12:1–12:46 (April 2010)
18. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Integrity for join queries in the cloud. IEEE TCC 1(2), 187–200 (July-December 2013)
19. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of ICDCS 2011. Minneapolis, MN, USA (June 2011)
20. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: Proc. of CRiSIS 2012. Cork, Ireland (October 2012)
21. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Protecting data in outsourcing scenarios. In: Das, S., Kant, K., Zhang, N. (eds.) Handbook on Securing Cyber-Physical Critical Infrastructure. Morgan Kaufmann (2012)
22. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Selective and fine-grained access to data in the cloud. In: Jajodia, S., Kant, K., Samarati, P., Swarup, V., Wang, C. (eds.) Secure Cloud Computing. Springer (2014)
23. Gamassi, M., Piuri, V., Sana, D., Scotti, F.: Robust fingerprint detection for access control. In: Proc. of RoboCare Workshop 2005. Rome, Italy (May 2005)
24. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. of STOC 2009. Bethesda, MA, USA (May 2009)
25. Goh, E.J.: Secure indexes. Tech. Rep. 2003/216, Cryptology ePrint Archive (2003), http://eprint.iacr.org/
26. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE 2002. San Jose, CA, USA (February 2002)
27. Hacigümüs, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Proc. of DASFAA 2004. Jeju Island, Korea (March 2004)
28. Jhawar, R., Piuri, V.: Fault tolerance management in IaaS clouds. In: Proc. of ESTEL 2012. Rome, Italy (October 2012)
29. Jhawar, R., Piuri, V.: Fault tolerance and resilience in cloud computing environments. In: Vacca, J. (ed.) Computer and Information Security Handbook, 2nd Edition, pp. 125–141. Morgan Kaufmann (2013)
30. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of CSE 2012. Paphos, Cyprus (December 2012)
31. Özsu, M., Valduriez, P.: Principles of distributed database systems. Prentice-Hall, Inc., 2 edn. (1999)

32. Samarati, P.: Data security and privacy in the cloud. In: Proc. of ISPEC 2014. Fuzhou, China (May 2014)
33. Schneier, B.: Applied Cryptography. John Wiley & Sons, 2/E (1996)
34. Wang, C., Cao, N., Ren, K., Lou, W.: Enabling secure and efficient ranked keyword search over outsourced cloud data. IEEE TPDS 23(8), 1467–1479 (August 2012)
35. Wang, H., Lakshmanan, L.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB 2006. Seoul, Korea (September 2006)
36. Winkler, V.: Securing the Cloud: Cloud Computer Security Techniques and Tactics. Syngress (2011)