Confidentiality Protection in Large Databases

Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Giovanni Livraga[1], Stefano Paraboschi[2], and Pierangela Samarati[1]

[1] Università degli Studi di Milano
via Bramante 65, 26013 Crema, Italy
firstname.lastname@unimi.it
[2] Università degli Studi di Bergamo
via Marconi 5, 24044 Dalmine, Italy
parabosc@unibg.it

Abstract. A growing trend in today's society is outsourcing large databases to the cloud. This permits to move the management burden from the data owner to external providers, which can make vast and scalable infrastructures available at competitive prices. Since large databases can include sensitive information, effective protection of data confidentiality is a key issue to fully enable data owners to enjoy the benefits of cloud-based solutions. Data encryption and data fragmentation have been proposed as two natural solutions for protecting data confidentiality. However, their adoption does not permit to completely delegate query evaluation at the provider. In this chapter, we illustrate some encryption-based and fragmentation-based solutions for protecting data confidentiality, discussing also how they support query execution.

1 Introduction

Starting from the pioneering Database-as-a-Service (DaaS) paradigm [20], recent years have seen an ever-growing trend towards the outsourcing of large data collections to external providers. By delegating data storage and management to external third parties, data owners can enjoy the immediate benefits of a reduced overhead at their side. The rapid advancements in cloud computing have accelerated this trend: the availability of a rich cloud market allows data owners to store and maintain huge data collections in the cloud at competitive (and typically pay-as-you-go) prices. The benefits of data outsourcing are multiple and not confined to economic factors, ranging from high service availability, to improved scalability and elasticity. However, no lunch comes for free, and one of the major issues arising when resorting to the cloud for data storage and management is the inevitable loss of control by the owner over her own data, and consequent risks to data protection in this context. Cloud providers are typically considered *honest-but-curious*, that is, trustworthy for correctly managing data but not trusted for accessing their content. Besides the intuitive need for protecting data confidentiality against external unauthorized subjects, it is then essential to protect it also against the provider itself.

Encryption represents a natural means for providing data confidentiality: by wrapping data with a layer of encryption, data are made unintelligible to unauthorized parties (which do not know the encryption key). If data are encrypted by their owner before being outsourced to the cloud, encryption also effectively provides data confidentiality against the storage provider itself [26]. Encrypting an entire data collection however can represent an overdo in many scenarios, where what is sensitive is the association among data items, rather than the data themselves. For instance, while knowing that a hospitalized patient is named Alice, and that a patient at the same hospital has flu might not be sensitive, the fact that Alice suffers from flu can be sensitive. The sensitive association between patients' names and diseases can be protected by simply storing names and diseases in two unlinkable data chunks, reducing the need for encryption. Following this intuition, the research community has proposed to combine encryption with data fragmentation [15]. In a nutshell, fragmentation-based techniques protect sensitive associations among data by splitting them into different unlinkable fragments (e.g., [1,5,7]).

Both encryption and fragmentation, while proved effective for confidentiality protection, cam impair query execution or make it more complex. Indeed, the cloud provider neither knows the encryption keys used to protect sensitive data nor can join fragments. Hence, it cannot evaluate user queries formulated on the original (plaintext and non-fragmented) relation. A naive solution would require the provider to return the entire encrypted or fragmented relation to the requesting user who (being authorized to issue queries) can decrypt or recombine it, to evaluate the query locally. This solution is however not viable, as it would nullify the benefits of resorting to the cloud. We will see how the use of indexes (metadata) associated with the encrypted relation or specific strategies allow the (partial) query evaluation directly at the provider, without requiring to decrypt data or join fragments (e.g., [1,5,7,20,13]).

In this chapter, we illustrate encryption-based and fragmentation-based techniques for protecting the confidentiality of large data collections outsourced at cloud providers, discussing also their support for query execution. Clearly, data confidentiality is only one aspect of the more general problem of ensuring proper protection to data. Many other problems have been investigated (e.g., [16,22,23]) but they are outside the scope of this chapter. The remainder of this chapter is organized as follow. Section 2 describes encryption-based solutions, while Section 3 presents fragmentation-based techniques, describing both the protection model and query evaluation approaches. Section 4 finally concludes the chapter.

2 Encryption-based Approaches

A natural solution for protecting data confidentiality consists in wrapping data with an *encryption* layer. Not knowing the encryption key, unauthorized subjects cannot decrypt the data and access their plaintext content. In this section, we discuss the use of encryption to protect the confidentiality of outsourced data, and illustrate different techniques that can be used to (partially) delegate query

FinancialData

SSN	Name	Race	Job	Salary	Ins
123-45-6789	Alice	white	teacher	40K	160
234-56-7890	Bob	while	farmer	25K	100
345-67-8901	Carol	asian	nurse	20K	100
456-78-9012	David	black	lawyer	50K	200
567-89-0123	Eric	black	secretary	20K	100
678-90-1234	Fred	asian	lawyer	40K	180

(a)

c_1={SSN}
c_2={Name, Salary}
c_3={Name, Ins}
c_4={Salary, Ins}
c_5={Race, Job, Ins}

(b)

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

evaluation on encrypted data to the cloud provider. We first focus on the index-based approach (Section 2.1), and then illustrate solutions that support the execution of queries directly over encrypted data (Section 2.2). For simplicity, we focus on outsourced data organized as a relation r defined over a relational schema $R(a_1, \ldots, a_n)$, with the note that the discussed protection techniques can also be applied to different (semi-)structured data models. We further note that the encryption schemas supporting keyword-based searches over generic encrypted data are outside the scope of this chapter and therefore we do not discuss them.

2.1 Encryption and Indexes

Data can be encrypted with symmetric as well as asymmetric encryption algorithms but, since symmetric encryption is typically cheaper, many proposals adopt symmetric encryption [26]. The outsourced relation can be encrypted at different granularity levels: cell level, attribute level, tuple level, or relation level. The chosen granularity level impacts on the query evaluation process, with consequences on its performance. For instance, relation-level encryption would require to return to the requesting user the entire encrypted relation. In general, finer granularity levels enable users to be more precise in downloading the encrypted content of interest but, on the other hand, cause a high overhead due to encryption/decryption operations. Viceversa, coarser granularity levels imply a lower overhead for encryption/decryption operations, but require to download larger encrypted chunks than necessary for query evaluation. Tuple-level encryption represents a good tradeoff between the overhead caused by encryption/decryption operations and query execution efficiency [26].

With tuple-level encryption, a relation r defined over relation schema $R(a_1, \ldots, a_n)$ is outsourced at the cloud provider as an encrypted relation r^e defined over schema $R^e(\underline{\texttt{tid}}, enc)$, with tid the primary key added to the encrypted relation and enc the encrypted tuple. Each tuple t in r is represented as an encrypted tuple t^e in r^e, where t^e[tid] is a random identifier and $t^e[enc]=E_k(t)$ is the encrypted tuple content, with E a symmetric encryption function with key k. To illustrate, consider relation FinancialData in Figure 1(a), reporting financial information about a set of individuals. Figure 2(a) illustrates an example of an encrypted version of it.

FinancialData^e

tid	enc
1	4tBf
2	lkG7
3	wF4t
4	m;Oi
5	n:8u
6	xF-g

(a)

FinancialData$^e_\iota$

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

(b)

Fig. 2: An example of encrypted (a) and encrypted and indexed (b) versions of relation FinancialData in Figure 1(a)

Query evaluation. To enable query evaluation over encrypted data at the cloud provider, without the need of decryption, index-based solutions complement the encrypted relation with indexes. Indexes are metadata that preserve some of the properties of the attributes on which they have been defined, and therefore can be used for query evaluation. Indexes are represented as additional attributes in the encrypted (and indexed) relation r^e_ι, which is then defined over schema $R^e_\iota(\underline{\texttt{tid}}, enc, I_{i_1}, \ldots, I_{i_j})$, with I_{i_l}, $l = 1, \ldots, j$ the index defined over attribute a_{i_l} in R. Note that not all attributes must be associated with an index – on the contrary, only those that are expected to be involved in conditions in query evaluation need to be indexed. Figure 2(b) illustrates an example of an encrypted and indexed version of relation FinancialData in Figure 1, with indexes over attributes `Race`, `Job`, `Salary`, and `Ins`. For simplicity, indexes are represented in the figure with Greek letters.

Different indexing techniques have been proposed, depending on the mapping between plaintext and index values and on the supported conditions [17,18]. Equality conditions of the form $a = v$, with v a value in the domain of a, are supported by many indexing techniques, such as *encryption-based* [13], *bucket-based* [20], and *hash-based* [13] indexes. Encryption-based indexes associate index value $E_k(t[a])$ with plaintext value $t[a]$, where E is a symmetric encryption scheme and k is the encryption key. Bucket-based indexes partition the domain of an attribute a in disjoint subsets of contiguous values each of which is associated with a *label*. A plaintext value $t[a]$ is represented in the index with the label l of the partition to which $t[a]$ belongs. Hash-based indexes instead adopt a secure hash function h that generates collisions, and the index value associated with plaintext value $t[a]$ is computed as $h(t[a])$. For instance, consider the encrypted and indexed relation in Figure 2: index I_r is an encryption-based index over attribute `Race` of relation FinancialData in Figure 1(a); index I_i is a partition-based index over attribute `Ins`, where the domain has been partitioned in two intervals: [100, 150] with label ν, and [151,200] with label ξ; and index I_j is a hash-based index over attribute `Job`, where the hash function is defined as follows: h(teacher)=h(farmer)=h(secretary)=δ and h(nurse)=h(lawyer)=ϵ.

Range conditions of the form a IN $[v_i, v_j]$, with a an attribute and $[v_i, v_j]$ a range in the domain of a are supported by bucket-based indexes, if the labels associated with them are defined so to preserve the order among attribute val-

q_u: SELECT Name	q_p: SELECT tid, enc
FROM *Decrypt*(R_p.*enc*,k)	FROM $\texttt{FinancialData}^e$
WHERE Job='teacher'	WHERE $\texttt{I}_r=\alpha$ AND $\texttt{I}_j=\delta$

Fig. 3: Execution of query "SELECT Name FROM FinancialData WHERE Race='white' AND Job='teacher' " over the encrypted and indexed relation of Figure 2(b) as subqueries at the provider side (q_p) and at the user side (q_u)

ues (leaking however the order of attribute values to the provider). An indexing technique specifically designed to support range queries is based on a $B+$-tree index defined over the plaintext attribute to be indexed [13]. The $B+$-tree index is represented at the provider through an additional encrypted relation. Alternative solutions to support range conditions rely on *Order Preserving Encryption Schemas* (OPES [2,28])

Aggregate operators [19,21] such as SUM and AVG are supported by indexes defined using homomorphic encryption [4,11,12,19,25], which allows the evaluation of arithmetic operators directly on encrypted data. The downside of these indexes is represented by the high computational overhead caused by homomorphic encryption schemes.

Given an encrypted and indexed relation r^e_ι over schema $R^e(\underline{\texttt{tid}}, enc)$, a query q formulated by the user on the original relation schema $R(a_1, \ldots, a_n)$ is translated into two queries q_p and q_u, operating at the provider and at the user sides, respectively. The original conditions in q are translated into equivalent conditions on the indexes in R^e to define the query q_p operating at the provider. Query q_u operates on the result of q_p to evaluate conditions that cannot be delegated to the provider (e.g., conditions over attributes that are not associated with indexes), and to filter the spurious tuples. For instance, with reference to the encrypted and indexed relation in Figure 2(b), Figure 3 illustrates an example of query execution over the encrypted and indexed relation in Figure 2(b).

2.2 Encrypted Data Processing

An alternative to indexes to partially delegate query execution to the provider consists in adopting encryption techniques that support the execution of operations or the evaluation of conditions directly over encrypted data. For instance, deterministic encryption supports the evaluation of equality conditions, Order Preserving Encryption (OPE) supports the evaluation of range conditions (e.g., [2,28]), and fully homomorphic encryption supports the evaluation of any function (e.g., [19]). Taking these encryption techniques as basic building blocks, some encrypted database systems have been developed (e.g., CryptDB [24], Monomi [27], and Cipherbase [3]), which support query processing over encrypted data. Among these systems, we describe CryptDB since it specifically focuses on query processing. Monomi focuses more on the physical design of the encrypted database with respect to a given workload and Cipherbase, being based on a trusted hardware on the untrusted cloud provider (which can per-

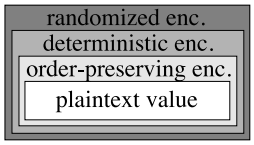

Fig. 4: An example of encryption layers adopted by CryptDB [24]

form arbitrary computation) focuses on how and where (client, server, trusted hardware) execute a computation.

CryptDB chooses a different encryption schema (depending on the conditions to be supported) for each attribute, and applies encryption at the cell level. To support different operations/conditions on the same attribute, multiple encryption layers are wrapped around a cell, forming an onion-like structure [24]. Note that the encryption layers are the same for all the cells in the same column, but they may vary from an attribute to another (depending on the kinds of queries to be supported). The outermost level features the strongest encryption (i.e., *randomized encryption*, a probabilistic scheme where two equal values are mapped onto different ciphertexts with non-negligible probability [24]), while the innermost level represents plaintext data. Proceeding from the outermost to the innermost layers, the adopted encryption scheme provides weaker security guarantees but supports more computations over encrypted data. For instance, attribute `Salary` in relation FINANCIALDATA in Figure 1 can be encrypted with randomized encryption (to maximize protection in storage), deterministic encryption (to support equality conditions), and OPE (to support aggregates and range conditions), as illustrated in Figure 4.

Encryption is dynamically regulated depending on the operations to be evaluated, by removing encryption layers. For instance, consider a query q : SELECT AVG(`Salary`) FROM `FinancialData`, and assume that the encryption layers are as in Figure 4. To enable the cloud provider to compute the average salary, the randomized and deterministic encryption layers are removed. Note that once a layer of encryption is removed from an attribute, it cannot be restored as data have been exposed to the provider, which cannot be considered oblivious.

Query evaluation. Query execution with CryptDB assumes a trusted proxy mediating the communications between the users and the cloud provider. The proxy stores a secret master key k, the database schema, and keeps track of the current encryption layers protecting each attribute in the relation. Given a query q issued by a user, the proxy rewrites it into an equivalent query $\hat{q}$ operating over the encrypted relation choosing, for each attribute on which the query operates, the most appropriate encryption layer in its onion structure. If one (or more) encryption layer(s) should be removed from one (or more) attribute(s), the proxy issues an UPDATE query removing it (them). The proxy forwards $\hat{q}$ to the cloud provider, which executes it. The provider then returns the encrypted result of $\hat{q}$ to the proxy, which finally decrypts it and sends the plaintext result to the user.

3 Fragmentation-based Approaches

Besides complicating query execution, encryption could also be an overdo since, in many scenarios, what is sensitive is the *association* among data (e.g., the identity of a patient and her salary), rather than the data items singly taken. In these cases, confidentiality of associations can be more conveniently protected by storing different portions of data in different, non-linkable fragments. In the context of fragmentation, sensitive attributes and sensitive associations among attributes can be formulated as *confidentiality constraints*, that is, sets of attributes whose joint visibility should be protected [1]. Singleton sets (*singleton constraints*) correspond to sensitive attributes; non-singleton sets (*association constraints*) correspond to sensitive associations. Figure 1(b) illustrates a set $\mathcal{C}$ of confidentiality constraints over relation FinancialData in Figure 1(a). In particular, c_1 is a singleton constraint stating that the `SSN` of the individuals is sensitive per se; $c_2, \ldots, c_5$ are association constraints stating that: the salary and insurance of an individual are sensitive (c_2 and c_3), the associations between a certain salary and the insurance (c_4), and among the race, job, and insurance of an individual (c_5) are sensitive. In this section, we illustrate three solutions that rely on fragmentation for protecting data confidentiality. We focus on solutions that assume attributes to be independent, while noting that fragmentation can also consider dependencies and correlations among attributes, which could introduce inferences channels [14].

3.1 Two can keep a secret

The first approach protecting data confidentiality through fragmentation guarantees data protection splitting the original relation into two fragments that do not include sensitive attributes or associations. The two fragments are guaranteed to be unlinkable as they are stored at two non-communicating providers [1].

Confidentiality constraints are satisfied by properly combining fragmentation and *encoding* [1]. Encoding an attribute a consists in splitting it into two attributes a^i and a^j, both necessary to reconstruct the values of a (i.e., $a = a^i \diamond a^j$, with $\diamond$ a non-invertible composition operator). Encryption represents a possible encoding technique, placing the ciphertext in a^i and the encryption key in a^j, with $\diamond$ the encryption algorithm. For simplicity, we will assume encryption as the adopted encoding technique.

A relation r over a schema R fragmented according to the proposal in [1] produces a fragmentation $\mathcal{F} = \{F_1, F_2, E\}$, with $F_1, F_2, E \subseteq R$. F_1 and F_2 are the sets of attributes represented in plaintext in the two fragments, while E is the set of encrypted attributes. Singleton confidentiality constraints are satisfied by encrypting the sensitive attributes (i.e., by placing them into E). Association constraints are typically satisfied by splitting the attributes they include between F_1 and F_2. Being the number of fragments fixed to two, it might happen that an attribute cannot be stored at any of the two fragments without violating some constraints. In such a situation, the attribute is encrypted. To illustrate, consider constraints c_2, c_3, and c_4 in Figure 1(b): it is not possible to split attributes

F_1^e

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

F_2^e

tid	Salary	SSN^2	Ins^2
1	40K	$k_{\texttt{SSN}}1$	$k_{\texttt{Ins}}1$
2	25K	$k_{\texttt{SSN}}2$	$k_{\texttt{Ins}}2$
3	20K	$k_{\texttt{SSN}}3$	$k_{\texttt{Ins}}3$
4	50K	$k_{\texttt{SSN}}4$	$k_{\texttt{Ins}}4$
5	20K	$k_{\texttt{SSN}}5$	$k_{\texttt{Ins}}5$
6	40K	$k_{\texttt{SSN}}6$	$k_{\texttt{Ins}}6$

Fig. 5: Two can keep a secret: An example of a correct fragmentation of relation FinancialData in Figure 1(a)

`Name`, `Salary`, and `Ins` between two fragments without violating a constraint. A fragmentation $\mathcal{F} = \{F_1, F_2, E\}$ is *correct* iff the following two conditions hold: *i)* $\forall c \in \mathcal{C} : c \not\subseteq F_1, c \not\subseteq F_2$ (*confidentiality*); and *ii)* $F_1 \cup F_2 \cup E = R$ (*completeness*). Condition *i)* ensures that neither F_1 nor F_2 store all the attributes in a confidentiality constraint in plaintext, while condition *ii)* ensures that all attributes of the original relation are included in the fragmentation (i.e., no information is lost due to fragmentation). A correct fragmentation guarantees that sensitive values and associations are not accessible to non-authorized users (since they do not know the encryption key, and the two providers do not communicate), and that the original relation r can be reconstructed by authorized users from $\mathcal{F}$. Considering the relation and the confidentiality constraints in Figure 1, an example of a correct fragmentation is $\mathcal{F} = \{F_1, F_2, E\}$, with $F_1 = \{\texttt{Name}, \texttt{Race}, \texttt{Job}\}, F_2 = \{\texttt{Salary}\}, E = \{\texttt{SSN}, \texttt{Ins}\}$.

At the physical level, fragments F_1 and F_2 are represented by two *physical fragments* F_1^e and F_2^e, where each physical fragment F_i^e stores the attributes in F_i in plaintext, a tuple identifier `tid`, and the encrypted attribute values (or the corresponding keys). The tuple identifier is needed for authorized users to recombine the content of the fragments, to reconstruct the original relation. 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 sharing the tuple identifier `tid`. Tuples t_1^e and t_2^e include in plaintext the values of the attributes in F_1 and F_2, respectively. Also, tuple t_1^e includes the encrypted values of the attributes in E, while t_2^e includes the corresponding encryption keys (or viceversa). Figure 5 illustrates the physical fragments F_1^e and F_2^e representing a correct fragmentation of relation FinancialData in Figure 1.

Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, there might exist different correct fragmentations satisfying $\mathcal{C}$. For instance, a fragmentation where fragments F_1 and F_2 are empty, and $E = R$ is clearly correct, but is typically undesirable since queries can be evaluated at the user side only. The authors of [1] then propose a metric to evaluate the quality of a fragmentation (where an optimal fragmentation minimizes the cost of evaluating a sample query workload on the user-side), and a heuristic algorithm to solve the (NP-hard) problem of computing such an optimal fragmentation.

Query evaluation. Since fragmentation should be transparent to final users, queries are formulated over the original relation r, and then are translated into a set of equivalent queries operating on the fragmentation $\mathcal{F}$. The intuition be-

Parallel strategy	Sequential strategy
q_1: SELECT tid, Name, Ins^1 FROM F_1^e WHERE Job='lawyer'	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 Salary=40K	q_2: SELECT tid, Ins^2 FROM F_2^e WHERE tid IN {4,6} AND Salary=40K
q_u: SELECT Name FROM R_1 JOIN R_2 ON R_1.tid=R_2.tid WHERE *Decrypt*(Ins^1, Ins^2)=180	q_u: SELECT Name FROM R_1 JOIN R_2 ON R_1.tid=R_2.tid WHERE *Decrypt*(Ins^1, Ins^2)=180

Fig. 6: Execution of query "SELECT Name FROM FinancialData WHERE Job='lawyer' AND Salary=40K AND Ins=180" over the fragments of Figure 5 as subqueries at the providers side (q_1 and q_2) and at the user side (q_u) with parallel and sequential strategies

hind such query translation process is that all conditions operating on attributes stored plaintext at F_1^e (F_2^e, resp.) can be easily evaluated by the provider storing F_1^e (F_2^e, resp.). All conditions over attributes in E or over pairs of attributes split between F_1 and F_2 cannot be evaluated by the providers; these conditions are evaluated at the user-side. Given a query q, it is then translated into three queries: q_1, which can be evaluated by the provider storing F_1^e; q_2, which can be evaluated by the provider storing F_2^e; and q_u, which must be evaluated by the user on the results of q_1 and q_2. Naturally, the translation must guarantee equivalence (i.e., the evaluation of q_1, q_2, and q_u must produce the same result as q), and it should push as much computation as possible to the providers to limit the burden at the user side. The query evaluation can follow two different strategies: the *parallel* strategy, where each provider evaluates its query and returns the result to the user, who joins them and evaluates query q_u to finally obtain the query result; and the *sequential* strategy where one of the two providers goes first and returns the result of its query to the user, who then passes the tuple identifiers in the query result to the second provider, and the user finally combines the two results of q_1 and q_2 and evaluates q_u. Both the parallel and the sequential strategies permit to correctly evaluate a query q on a fragmentation $\mathcal{F} = \{F_1, F_2, E\}$. The parallel strategy, while reducing response time (the providers execute their queries at the same time), is likely to cause a higher communication costs. The sequential strategy may instead cause a delay in obtaining the final query result since the results from a provider are needed before a query is sent to the other provider.

Figure 6 illustrates an example of query execution over the fragments in Figure 5. Here, condition tid IN $\{4, 6\}$ in q_2 of the sequential strategy is needed to consider in the execution of q_2 only the tuples returned by q_1.

3.2 Multiple fragments

Two can keep a secret approach guarantees the unlinkability of fragments, needed to ensure satisfaction of constraints, by assuming the two providers to neither communicate nor collude. However, this assumption is difficult to enforce in practice, and thus reduces the applicability of this model in real world scenarios. The *multiple fragments* approach overcomes this assumption by defining an arbitrary number of uninkable fragments [5].

Unlinkability among fragments is guaranteed by requiring that no plaintext attribute is included in more than one fragment. If each fragment singly taken satisfies the constraints, then all fragments of a fragmentation can be stored at the same provider without risks for confidentiality.

Confidentiality constraints are satisfied (like in the proposal in [1]) by properly combining fragmentation and encryption. More precisely, singleton constraints are satisfied by encrypting the attribute they involve. Association constraints can instead be satisfied by either encrypting at least one of the attributes they include (satisfaction through encryption), or storing these attributes in different fragments (satisfaction through fragmentation). A fragmentation $\mathcal{F} = \{F_1, \ldots, F_m\}$ is *correct* iff the following two conditions hold: *i)* $\forall c \in \mathcal{C}$, $\forall F \in \mathcal{F}$: $c \not\subseteq F$ (*confidentiality*); *ii)* $\forall F_i, F_j \in \mathcal{F}$, $i \neq j$: $F_i \cap F_j = \emptyset$ (*unlinkability*). Condition *i)* ensures that no fragment in $\mathcal{F}$ can contain all the attributes included in a confidentiality constraint. Condition *ii)* ensures instead that all fragments in the fragmentation are disjoint. For instance, $\mathcal{F}$={{`Name`,`Job`}, {`Salary`}, {`Race`,`Ins`}} is a correct fragmentation of relation FinancialData in Figure 1(a) with respect to the constraints in Figure 1(b).

The multiple fragments approach has two immediate advantages over the solution in [1]. First, the entire fragmentation can be stored at a single cloud provider (as fragments $F_1, \ldots, F_n$ of $\mathcal{F}$ are disjoint), thus removing the need for having multiple non-communicating providers. Second, being the fragmentation not limited to two fragments, it is always possible to satisfy an association constraint through fragmentation, hence limiting encryption to the satisfaction of singleton constraints only. Such an approach increases the *visibility* over the data, improving the performance in accessing data (e.g., for query evaluation): the plaintext inclusion of an attribute a in a fragment F allows for the evaluation of conditions over a directly at the cloud provider storing F. The multiple fragments approach aims therefore at computing fragmentations that *maximize plaintext visibility*, that is, where each attribute not included in a singleton constraint is plaintext represented in *at least* one fragment. Note that, combining this requirement with the unlinkability condition, each attribute not involved in a singleton constraint is plaintext included in *exactly* one fragment. For instance, $\mathcal{F}$={{`Name`,`Job`}, {`Salary`}, {`Race`,`Ins`}} is a correct fragmentation of relation FinancialData in Figure 1 that maximizes visibility.

At the physical level, each fragment F_i is represented by a *physical fragment* F_i^e storing: the attributes in F_i in plaintext; the attributes in $R \setminus F_i$ in encrypted form; and a primary key `salt` containing random values. Each tuple t in r is represented by a tuple t_i^e in each physical fragment F_i^e, where $t_i^e[a]$=$t[a]$ for all

F_1^e

salt	enc	Name	Job
s_{11}	xTb:	Alice	teacher
s_{12}	o;!G	Bob	farmer
s_{13}	Ap'L	Carol	nurse
s_{14}	.u7t	David	lawyer
s_{15}	y"e3	Eric	secretary
s_{16}	(l1!	Fred	lawyer

F_2^e

salt	enc	Salary
s_{21}	hg5=	40K
s_{22}	mB71	25K
s_{23}	:k?2	20K
s_{24}	Ql4,	50K
s_{25}	-kGd	20K
s_{26}	p[Mz	40K

F_3^e

salt	enc	Race	Ins
s_{31}	bP5	white	160
s_{32}	*Cx	white	100
s_{33}	1Bny	asian	100
s_{34}	Oj)6	black	200
s_{35}	vT7/	black	100
s_{36}	l1fY	asian	180

Fig. 7: Multiple fragments: An example of a correct fragmentation of relation FINANCIALDATA in Figure 1(a)

q_u: SELECT `Name`
FROM *Decrypt*(R_p.`enc`⊕`salt`,k)
WHERE `Salary`=40K

q_p: SELECT `salt`, `enc`, `Name`
FROM F_1^e
WHERE `Job`='teacher'

Fig. 8: Execution of query "SELECT `Name` FROM `FinancialData` WHERE `Salary`=40K AND `Job`='teacher' " over fragment F_1^e of Figure 7 as subqueries at the provider's side (q_p) and at the user's side (q_u)

$a \in F_i$; t_i^e[`salt`] is a random nonce; and t_i^e[`encr`] $= E_k(t[a_j, \ldots, a_k] \oplus t^e$[`salt`]), with $\{a_j, \ldots, a_k\} = R \setminus F_i$. Since each physical fragment stores, either plaintext or encrypted, all the attributes in R, every query can be evaluated on a single fragment. Figure 7 illustrates the physical fragments of a correct fragmentation of the relation in Figure 1(a) with respect to the constraints in Figure 1(b).

Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, there might exist different correct fragmentations (i.e., satisfying $\mathcal{C}$) that maximize visibility. For instance, a fragmentation where all attributes not involved in singleton constraints are stored at a different fragment would be correct but likely undesirable, complicating the execution of queries involving more than one attribute. Different metrics have therefore been proposed to evaluate the quality of a fragmentation $\mathcal{F}$, aimed at minimizing the number of fragments in $\mathcal{F}$ (e.g., [5,10,14]), or the cost needed to execute a query workload (e.g. [6,8]).

Query evaluation. Since all physical fragments of a fragmentation include, either plaintext or encrypted, all the attributes of R, a query q can be executed over any fragment. However, from a user's point of view, it is clearly more convenient to use the fragment that permits to delegate to the cloud provider most of the query evaluation (i.e., the fragment storing in plaintext most of the attributes in which the conditions of q operate). The original query is then translated into an equivalent pair of queries q_p and q_u, operating at the provider and at the user sides, respectively. Query q_p, which can include conditions over the attributes plaintext represented in the fragment, is sent to the provider, which returns the result to the user. The user decrypts the encrypted attributes in the result of q_p (if any), and evaluates q_u, evaluating conditions over the attributes not plaintext included in the fragment. With reference to the relation in Figure 1(a) and the

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_p^e

tid	Race	Job	Salary
1	white	teacher	40K
2	while	farmer	25K
3	asian	nurse	20K
4	black	lawyer	50K
5	black	secretary	20K
6	asian	lawyer	40K

Fig. 9: Keep a few: An example of a correct fragmentation of relation FINANCIALDATA in Figure 1(a)

fragmentation in Figure 7, Figure 8 illustrates an example of query execution, assuming to choose fragment F_1^e.

3.3 Keep a few

While both the two can keep a secret and the multiple fragments approaches build upon a combination of fragmentation and encryption, the *keep a few* approach [7,9] completely departs from encryption. Such an approach can be adopted when the data owner (or a trusted third party) is available for storing a limited portion of the data. Confidentiality constraints are then satisfied by combining fragmentation with owner-side storage. More precisely, singleton constraints are satisfied by storing at the owner side the attribute they involve. Similarly, association constraints are satisfied by storing at least one of the attributes they include at the owner side. A fragmentation $\mathcal{F}$ of a relation r defined over relation schema R is then a pair $\mathcal{F} = \langle F_o, F_p \rangle$ of fragments, with $F_o, F_p \subseteq R$, and where F_o is stored at the data owner and F_p is outsourced at a cloud provider. A fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ is correct iff the following two conditions hold: *i)* $\forall c \in \mathcal{C}, c \not\subseteq F_p$ (*confidentiality*); *ii)* $F_o \cup F_p = R$ (*losslessness*). Condition *i)* ensures that F_p does not contain all the attributes involved in a confidentiality constraint. Note that the confidentiality condition is not needed to hold on F_o, since this fragment is stored at the owner and hence is not accessible to non-authorized users. Condition *ii)* ensures that the fragmentation includes all the attributes in R, guaranteeing that no information is lost due to fragmentation. Note that, since including the same attribute in both F_o and F_p would be redundant, the two fragments are typically required to be disjoint (i.e., $F_p \cap F_o = \emptyset$). For instance, $\mathcal{F} = \langle F_o, F_p \rangle$ with F_o={`SSN`,`Name`,`Ins`} and F_p={`Race`,`Job`,`Salary`} is an example of a correct and non-redundant fragmentation of relation FINANCIALDATA in Figure 1(a), with respect to the confidentiality constraints in Figure 1(b).

At the physical level, fragments F_p and F_o are translated into two *physical fragments* F_p^e and F_o^e, including all attributes (plaintext, as no encryption is adopted with this approach) in F_p and F_o, respectively. Both physical fragments are also enriched with a common tuple identifier `tid` that authorized users can use to correctly reconstruct the original relation. Figure 9 illustrates the physical fragments of a correct fragmentation of relation FINANCIALDATA in Figure 1(a).

Provider-Owner strategy	Owner-Provider strategy
q_p: SELECT `tid`, `Name` FROM F_p^e WHERE `Salary`=20K q_o: SELECT `Name` FROM F_p^e JOIN R_p ON F_p^e.`tid`=R_p.`tid` WHERE `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 `Salary`=20K q_{po}: SELECT `Name` FROM F_o^e JOIN R_p ON F_o^e.`tid`=R_p.`tid`

Fig. 10: Execution of query "SELECT `Name` FROM `FinancialData` WHERE `Salary`=20K AND `Ins`=180" over the fragments of Figure 9 as subqueries at the provider side (q_p) and at the owner side (q_o and q_{po}), with provider-owner and owner-provider strategies

Given a relation schema R and a set $\mathcal{C}$ of confidentiality constraints over it, there can exist different correct and non-redundant fragmentations. For instance, a fragmentation $\mathcal{F} = \langle F_o, F_p \rangle$ where $F_p = \emptyset$ and $F_o = R$ is correct and non-redundant, but likely to be undesirable since no attribute is outsourced to the cloud. Several metrics have been proposed to evaluate the quality of a fragmentation, aimed at minimizing the burden for the data owner, which can be measured in terms of the number/size of the attributes stored owner-side at F_o, or of the computational overhead left at the data owner based on a query workload [7,9].

Query evaluation. A query q formulated over the original relation schema R is transformed into two equivalent queries q_p and q_o, operating at the provider and at the owner sides, respectively: q_p includes conditions operating only on attributes in F_p; q_o includes conditions operating on attributes in F_o (or comparing attributes in the two fragments). The evaluation of q can follow two strategies, depending on the order in which q_p and q_o are evaluated. In the *provider-owner* strategy, the provider evaluates q_p and returns the result to the owner, who joins it with its fragment and evaluates q_o to obtain the result of q. In the *owner-provider* strategy, the owner evaluates conditions in q that involve only attributes in F_o, and sends then the identifier of the tuples satisfying such conditions to the provider. The provider evaluates q_p on these tuples, and returns the result to the owner. The data owner joins her fragment with the result computed by the provider and evaluates the conditions that involve attributes in both fragments, obtaining the result of q. Figure 10 illustrates an example of query execution according to these two strategies.

While both the provider-owner strategy and the owner-provider strategy correctly compute the result of a query q, the latter can leak sensitive information. In fact, if the provider knows the original query q, it can learn which are the tuples in F_p that satisfy the conditions evaluated by the data owner (the individ-

uals with `Ins` equal to 100 in our example), even if the provider is not authorized to see the content of the attributes in F_o.

4 Conclusions

We have illustrated encryption-based and fragmentation-based solutions for protecting confidentiality in large databases when they are outsourced to the cloud. Since both encryption and fragmentation complicate or even prevent query execution at the provider, we have also illustrated some of the existing approaches that enable (partial) query execution directly at the provider, without the need for decrypting encrypted data, or of joining fragments.

Acknowledgments This work was supported in part by the EC within the H2020 under grant agreement 644579 (ESCUDO-CLOUD), and within the FP7 under grant agreement 312797 (ABC4EU).

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. Asilomar, CA, USA (January 2005)
2. Agrawal, R., Kierman, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of SIGMOD. Paris, France (June 2004)
3. Arasu, A., Blanas, S., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R., Upadhyaya, P., Venkatesan, R.: Secure database-as-a-service with cipherbase. In: Proc. of SIGMOD 2013. New York, New York, USA (June 2013)
4. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing 43(2), 831–871 (April 2014)
5. 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. Dresden, Germany (September 2007)
6. 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. Montreal, Canada (June 2009)
7. 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. Saint Malo, France (September 2009)
8. 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)
9. 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)
10. 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)

11. Coron, J.S., Mandal, A., Naccache, D., Tibouchi, M.: Fully homomorphic encryption over the integers with shorter public keys. In: Proc. of CRYPTO. Santa Barbara, CA, USA (August 2011)
12. Coron, J.S., Naccache, D., Tibouchi, M.: Public key compression and modulus switching for fully homomorphic encryption over the integers. In: Proc. of EUROCRYPT. Cambridge, UK (April 2012)
13. 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. Washington, DC, USA (October 2003)
14. 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
15. De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Practical techniques building on encryption for protecting and managing data in the cloud. In: Ryan, P., Naccache, D., Quisquater, J.J. (eds.) Festschrift for David Kahn. Springer (2016)
16. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: Proc. of CRiSIS. Cork, Ireland (October 2012)
17. 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)
18. 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)
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. of STOC. Bethesda, MA, USA (May 2009)
20. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE. San Jose, CA, USA (February 2002)
21. Hacigümüs, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Proc. of DASFAA. Jeju Island, Korea (March 2004)
22. 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)
23. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of CSE. Paphos, Cyprus (December 2012)
24. Popa, R., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting confidentiality with encrypted query processing. In: Proc. of SOSP. Cascais, Portugal (October 2011)
25. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: DeMillo, R., Lipton, R., Jones, A. (eds.) Foundation of Secure Computations. Academic Press (1978)
26. Samarati, P., De Capitani di Vimercati, S.: Cloud security: Issues and concerns. In: Murugesan, S., Bojanova, I. (eds.) Encyclopedia on Cloud Computing. Wiley (2016)
27. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. Proceedings of the VLDB Endowment 6(5), 289–300 (2013)
28. Wang, H., Lakshmanan, L.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB. Seoul, Korea (September 2006)