Chapter 5.1 – Protecting Data in Outsourcing Scenarios

Sabrina De Capitani di Vimercati, Sara Foresti, Pierangela Samarati

DTI - Università degli Studi di Milano
via Bramante, 65 - 26013 Crema, Italy
firstname.lastname@unimi.it

Abstract

Goal of this chapter is to describe the main solutions being devised for protecting data confidentiality and integrity in outsourcing scenarios. In particular, we illustrate approaches that guarantee data confidentiality by applying encryption or a combination of encryption and fragmentation. We then focus on approaches that aim at guaranteeing data integrity in storage and in query computation. Finally, we present some issues that still need to be investigated for ensuring privacy and security of data outsourced to external servers.

keywords: Data outsourcing, index, selective encryption, confidentiality constraint, fragmentation, integrity

1 Introduction

The rapid advancements in Information and Communication Technologies (ICTs) have brought to the development of new computing paradigms, where the techniques for processing, storing, communicating, sharing, and disseminating information have radically changed. Individuals and organizations are more and more resorting to external servers [28] for the storage and the efficient and reliable dissemination of information. The main consequence of this trend is the development of a *data outsourcing* architecture on a wide scale. While data outsourcing introduces many benefits in terms of management and availability of information, it also introduces

new privacy and security concerns. In fact, data are no more under the direct control of their owners, and their confidentiality and integrity may then be put at risk. The need for a proper protection of outsourced data is also exacerbated by the sensitive nature of the information collected and stored at external servers. Outsourced data often include sensitive information (e.g., identifying information [8, 21, 22], financial data, health diagnosis) whose protection is mandatory. Individuals as well as companies require the protection of their sensitive information not only against external users breaking into the system but also against malicious insiders. In many cases, the external server is relied upon for ensuring availability of data but should not be allowed to read the data it stores. Protection must therefore be also ensured against possible *honest-but-curious* servers, that is, servers that honestly manage data but may not be trusted by the data owner to read their content. Ensuring effective and practical data protection in such contexts is a complex task that requires to design effective approaches allowing data owners to specify privacy requirements on data, and techniques for enforcing such requirements in data storage and processing. Data privacy and integrity issues in outsourcing scenarios have then captured the attention of the research community and several advancements have been proposed (e.g., see [40]). The main goal of this chapter is to investigate different approaches that have been proposed by the research community for protecting the confidentiality and integrity of outsourced data. In particular, we consider a scenario where data are stored in relational databases. The techniques discussed can however be adopted for protecting any kind of data.
The remainder of this chapter is organized as follows. Section 2 describes different approaches for efficiently evaluating queries and enforcing access control policies over outsourced encrypted data. Encryption, however, makes access to stored data inefficient because it is not always possible to directly evaluate queries on encrypted data. Section 3 then describes alternative solutions that grant data confidentiality and efficient query evaluation. These approaches model privacy requirements through confidentiality constraints, and enforce them by combining fragmentation and encryption, and possibly involving the data owner for storing a limited portion of sensitive data. Section 4 describes the main approaches for protecting data integrity in

storage and query computation. Section 5 presents some open issues that are currently under investigation by the research community for providing data privacy and integrity and for supporting complex protection requirements in outsourcing scenarios. Finally, Section 6 concludes the chapter.

2 Data encryption

A possible solution for protecting outsourced data from the server storing them consists in applying encryption. Encryption protects the exposure of sensitive information even if the server is compromised and ensures integrity since data tampering can be detected. Clearly, data decryption cannot be executed at the server side, and therefore solutions have been developed that allow the external server to execute queries directly on the encrypted data. In the remainder of this section, we first describe how data are stored at the external server and then illustrate how query execution and access control are enforced.

2.1 Data organization

Data outsourcing typically involves four parties: a *data owner* is an organization (or an individual) who outsources her data to make them available for controlled external release; a *user* is a person who can access the outsourced data; a *client* is the user's front-end, which is in charge of translating access requests formulated by the user in equivalent requests operating on the outsourced data; a *server* is the external third party that stores and manages the outsourced data. We assume that the outsourced data are stored in a relational database. Before outsourcing a plaintext database $\mathcal{B}$, each relation in the database is mapped to an encrypted relation. In principle, data can be encrypted by using either a symmetric or an asymmetric encryption function. However, since symmetric encryption is cheaper than asymmetric encryption, solutions are typically based on symmetric encryption (e.g., [15, 24]). Data encryption can be applied at different granularity levels: table, attribute, tuple, or cell. Table and attribute level encryption

PATIENTS

SSN	Name	DoB	County	Diagnosis	Prescription
782619730	Anne	55/01/23	Alameda	HIV	Nevirapine
946294626	Beth	86/04/05	Fresno	Anemia	Folic acid
737260262	Cheryl	40/12/23	Napa	Arthritis	Anti-inflammatory
937360965	Doris	81/07/22	Napa	Diabetes	Insulin
946259572	Evelyn	65/10/03	Orange	HeartAttack	Anticoagulants
837350362	Flora	89/03/24	Trinity	Diabetes	Insulin

(a)

PATIENTS^k

tid	etuple	$\mathbf{I}_n$	$\mathbf{I}_b$	$\mathbf{I}_c$	$\mathbf{I}_d$
1	zKZlJxV	α	ϕ	υ	θ
2	AJvaAy1	β	ς	ι	κ
3	AwLBAa1	γ	χ	λ	μ
4	mHF/hd8	δ	ς	η	π
5	HTGhoAq	ϵ	ϕ	υ	ρ
6	u292mdo	ζ	ς	σ	π

(b)

Figure 1: An example of plaintext relation (a) and the corresponding encrypted relation (b)

imply that the whole relation involved in a query should always be returned since the server cannot select the data of interest, thus leaving to the client the burden of executing a query on a potentially huge amount of data. Encryption at the cell level implies an excessive workload for the client that needs to execute a possibly very large number of decrypt operations. For these reasons, many proposals adopt encryption at the tuple level, since it represents a good trade-off between encryption/decryption workload and query execution efficiency.

To directly query encrypted data, a set of *indexes* (e.g., [15, 24, 27]) are typically stored with the encrypted relation. Indexes, whose values are computed starting from the plaintext values of the attributes with which they are associated, allow the server storing the data to partially evaluate clients' queries (see Section 2.2). Each relation r in $\mathcal{B}$ defined over schema $R(A_1, \ldots, A_n)$ is then mapped to an encrypted relation r^k over schema $R^k(\underline{tid}, etuple, I_{i_1}, \ldots, I_{i_j})$ in the encrypted database $\mathcal{B}^k$ stored at the external server, with *tid* a numerical attribute added to the encrypted relation and acting as a primary key for R^k; *etuple* the encrypted tuple; $I_{i_l}, l = 1, \ldots, j$, index associated with the i_l-th attribute A_{i_l} in R on which conditions need to be evaluated in the execution of queries. Each tuple t in r is mapped to a tuple t^k in r^k, where $t^k[etuple] = E_k(t)$ and $t^k[I_{i_l}] = f(t[A_{i_l}]), l = 1, \ldots, j$, where E is an encryption function, k is an encryption key, and f is an indexing function. The resulting encrypted relation r^k contains the same number of tuples as the plaintext relation r, since the number of tuples is not affected by encryption and indexing.

Example 2.1. *Consider plaintext relation* PATIENTS *in Figure 1(a) and suppose that there*

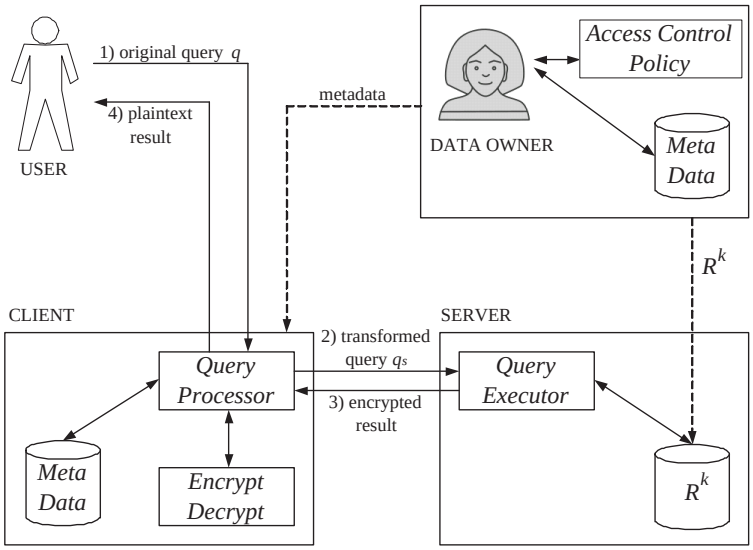


Figure 2: Query evaluation process

is an index for attributes `Name` *(*I_n*),* `DoB` *(*I_b*),* `County` *(*I_c*), and* `Diagnosis` *(*I_d*). Figure 1(b) illustrates the corresponding encrypted relation* PATIENTS^k*, where index values are represented with Greek letters. For readability, we report the tuples in the plaintext and encrypted relations in the same order. Note, however, that the order in which tuples are stored in the encrypted relation is independent from the order in which they appear in the plaintext relation.*

2.2 Query execution

The introduction of indexes allows the server to partially evaluate a query q submitted by the client. Figure 2 illustrates the query evaluation process. The original query q formulated by the user (who may not be aware of the fact that data have been outsourced) on the plaintext relation is sent to a trusted client (step 1). The client maps q into two queries: q_s operates on the encrypted relation using indexes, and q_c operates on the result of q_s. Query q_s is then communicated to the external server (step 2). The external server executes query q_s on the encrypted relation and returns the result to the client (step 3). The client decrypts the result obtained from the server, and evaluates q_c on the resulting relation to possibly remove *spurious tuples* (i.e., tuples that do not belong to the final result), and returns the result to the user

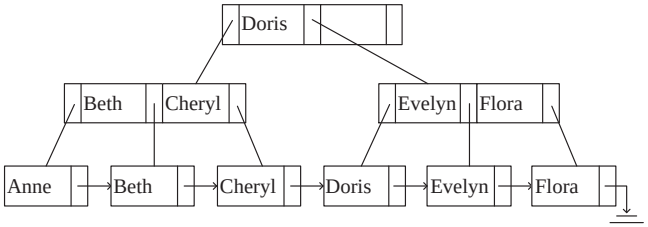
(step 4). Clearly, the translation of query q into queries q_s and q_c depends on the kind of indexes involved in the query since different indexes support different queries (e.g., some indexes do not support range queries and therefore they have to be executed at the client side). In the following, we briefly describe the main indexing techniques.

- *Encryption-based indexes* (e.g., [15]). A simple approach consists in using as index the result of an encryption function over the actual values. Given a tuple t in r, the value of index I_i, associated with attribute A_i for tuple t, is computed as $E_k(t[A_i])$, where E_k is a symmetric encryption function and k the encryption key. This indexing technique has the advantage of supporting equality queries, that is, queries with conditions of the form $A_i{=}v$ and $A_i{=}A_j$. In fact, these conditions can be translated into conditions $I_i{=}E_k(v)$ and $I_i{=}I_j$, respectively, operating on the encrypted relation. Note that the index values for attributes A_i and A_j must be computed using the same encryption function with the same key. For instance, suppose that index I_n in Figure 1(b) has been obtained by adopting this method. Then, with reference to the plaintext and encrypted relations in Figure 1, condition `Name` LIKE 'Evelyn' will be translated as $I_n{=}$'ϵ' on relation PATIENTS^k. Since encryption-based indexes preserve plaintext distinguishability, all the tuples returned by the server belong to the result of the original query. As a drawback, this technique does not easily support range queries, because encryption functions are not order preserving. We note however that a range condition can be translated into a set of equality conditions, one for each of the values in the range. For instance, a range condition `Name` LIKE '[A–B]%' is translated as $I_n{=}$'α' OR $I_n{=}$'β'.

- *Partition-based indexes* (e.g., [27]). The domain D_i of attribute A_i is partitioned into a set of non-overlapping subsets of contiguous values, which are usually of the same size. Each partition is associated with a label that may or may not preserve the order relationship characterizing values in D_i. Given a tuple t in r, the value of index I_i, associated with attribute A_i for tuple t, is the label of the unique partition containing value $t[A_i]$. For instance, index I_b in Figure 1(b) is obtained by partitioning the domain [1910-

01-01,2010-12-31] of attribute DoB in intervals of 20 years, and assigning, in the order, labels ω, χ, ϕ, ς, and τ to the resulting partitions. This indexing method allows the server side evaluation of equality conditions, by translating a condition of the form A_i=v into condition I_i=*partition*(v), where *partition*(v) is the label of the partition including value v. Like for encryption-based indexes, this indexing technique supports the evaluation of equality conditions between two attributes (i.e., A_i=A_j), provided that the attributes are defined on the same domain and they have been indexed using the same partition. Since an index value (i.e., the label associated with a partition) corresponds to different plaintext values (i.e., all plaintext values belonging to the partition), the query result computed by the server can include spurious tuples that need to be eliminated by the client. For instance, with reference to the plaintext and encrypted relations in Figure 1, condition DoB='86/04/05' is translated as I_b='ς'. The server therefore returns the second, fourth, and sixth encrypted tuple. The client decrypts these three tuples, and eliminates the latter two (whose presence is due to index collision) by reevaluating the condition on plaintext data. Partition-based indexes do not easily support range queries. In fact, if the index values are not order-preserving, a condition of the form $A_i \leq v$ is translated into condition $(I_i = idx_1) \vee \ldots \vee (I_i = idx_m)$, where each value $idx_j, j = 1, \ldots, m$, is the label of a partition including plaintext values that are lower than or equal to v. For instance, condition DoB$\leq$'83/12/31' is translated as I_b='ω' OR I_b='χ' OR I_b='ϕ' OR I_b='ς'.

- *Hash-based indexes* (e.g., [15]). Given a tuple t in r, the value of index I_i, associated with attribute A_i for tuple t, is computed as $h(t[A_i])$, where h is a hash function. The hash function satisfies three properties: *i)* it is deterministic, meaning that given two values v_1 and v_2 in the domain D_i of attribute A_i, if $v_1 = v_2$ then $h(v_1) = h(v_2)$; *ii)* it generates collisions that happen when given two values v_1 and v_2 in the domain D_i of attribute A_i, with $v_1 \neq v_2$, $h(v_1) = h(v_2)$; and *iii)* it is not order preserving. A hash-based index allows the server side evaluation of equality conditions of the form A_i=v and A_i=A_j that are translated as I_i=$h(v)$ and as I_i=I_j, respectively (provided the hash function adopted to



(a)

id_node	node
1	2, Doris, 3
2	4, Beth, 5, Cheryl, 6
3	7, Evelyn, 8, Flora, 9
4	Anne, 5, t_1
5	Beth, 6, t_2
6	Cheryl, 7, t_3
7	Doris, 8, t_4
8	Evelyn, 9, t_5
9	Flora, NIL, t_6

(b)

id	content
1	8/*5sym,p
2	mw39wio[
3	gtem945/*c
4	gte3/)8*
5	8dq59wq*d'
6	ue63/)w
7	=wco21!ps
8	oieb5(p8*
9	21!p8dq59

(c)

Figure 3: An example of B+tree index (a), its relational representation (b), and the corresponding encrypted relation (c)

define I_i is the same used to define I_j). For instance, suppose that index I_c in Figure 1(b) has been obtained by adopting a hash-based indexing method. With reference to the plaintext and encrypted relations in Figure 1, condition `County`='Fresno' is translated as I_c=h(Fresno)='ι'. Since the hash function is not order preserving, hash-based indexes do not easily support range queries.

- *B+-tree indexes* (e.g., [15]). A B+-tree data structure is used for indexing data. The B+-tree index is built by the data owner over the original plaintext values of an attribute A_i. The B+-tree is then encrypted at node level (i.e., each node of the B+-tree is encrypted as a whole) and stored at the server as a table with two attributes: *id* contains the node identifier, and *content* contains an encrypted value representing the node content. Pointers to children are represented through cross references from the node content to the node identifiers of its children in the table. For instance, Figure 3(a) illustrates the B+-tree index built for attribute `Name` of relation PATIENTS in Figure 1(a). Figure 3(b) illustrates the relation representing the B+-tree in Figure 3(a), and Figure 3(c) illustrates the corresponding encrypted relation stored at the external server. B+-tree indexes support both equality and range queries and, being order preserving, allow the server to evaluate GROUP BY and ORDER BY SQL clauses. Since the B+-tree index is encrypted, the traversal of the index can only be performed by the client. To execute

a range query, the client has to perform a sequence of queries that retrieve tree nodes at progressively deeper levels; when a leaf is reached, the node identifiers in the leaf can be used to retrieve the tuples belonging to the interval. For instance, with reference to the plaintext relation in Figure 1(a) and the index structure in Figure 3, condition `Name` LIKE '[D–Z]%' (retrieving the names following 'D' in lexicographic order) is evaluated by executing a set of queries for traversing the $B+$-tree along the path of nodes 1, 3, and 7. Then, other queries will be produced to traverse the chain of leaves starting from node 7. For each visited leaf, the client also retrieves the tuples associated with it (i.e., tuple t_4 for leaf 7, tuple t_5 for leaf 8, and tuple t_6 for leaf 9).

- *Order preserving encryption indexes* (e.g., [2, 45]). Alternative approaches that support equality and range queries are the *Order Preserving Encryption Schema* (OPES) [2] and the *Order Preserving Encryption with Splitting and Scaling* (OPESS) schema [45]. OPES is an encryption technique that takes as input a target distribution of index values and applies an order preserving transformation in a way that the transformed values (i.e., the index values) follow the target distribution. Comparison operations can be directly applied on the encrypted data, thus avoiding the production of spurious tuples. OPESS adopts splitting and scaling techniques to create index values so that the frequency distribution of index values is flat (i.e., uniform).

- *Privacy homomorphic indexes* (e.g., [23, 26]). Although order preserving encryption techniques permit the efficient evaluation of comparison operations, they do not support arithmetic operations. As a consequence, the evaluation of aggregate functions (e.g., SUM, AVG) cannot be delegated to the server. Privacy homomorphic encryption [38] allows the execution of basic arithmetic operations (i.e., $+$,$-$,$\times$) over encrypted data. Their adoption in the definition of indexes can then allow the server to directly evaluate aggregate functions as well as execute equality and range queries [26]. As a drawback, the computation of arithmetic operations over encrypted data is time consuming. Recently, in [23] a fully

homomorphic encryption schema has been proposed that supports the computation of an arbitrary function over encrypted data without the decryption key. While this schema represents an important theoretical result, it cannot be used in real-world scenarios due to its exponential computational complexity.

Besides the main indexing techniques illustrated above, many other solutions have been proposed that aim at better supporting SQL clauses or at reducing the client workload in the query evaluation process (e.g., [47]).

We conclude this overview on indexing techniques observing that the definition of an index over an attribute must consider two conflicting requirements: on one hand, the index values should be related to the corresponding plaintext data well enough to provide for an effective query execution process; on the other hand, the relationship between indexes and plaintext data should not open the door to inference and linking attacks. Different indexing methods can provide a different trade-off between query execution efficiency and data protection from inference. For instance, with an encryption-based index, index values reproduce exactly the plaintext values distribution, thus opening the door to frequency-based attacks. An analysis of the risk of exposure due to the publication of indexes is therefore an important aspect that however only few proposals have considered. Also, as demonstrated in [7], even a limited number of indexes can greatly facilitate the task for an adversary who wants to violate the confidentiality provided by encryption.

2.3 Access control enforcement

The majority of the solutions developed in the data outsourcing scenario focus on the definition of indexing techniques (e.g., [15, 24, 27]) and therefore no attention is posed on how data are encrypted. Typically, data are assumed to be encrypted with a key that is shared among all users that can access the data. As a consequence, all users knowing the encryption key can access the whole outsourced database. This situation is clearly limiting in a real world scenario, where different users may instead have different access privileges. However, the enforcement of

	t_1	t_2	t_3	t_4	t_5	t_6
G	1	1	1	1	0	0
H	1	1	1	1	1	0
I	1	1	1	0	1	1
J	0	0	1	1	1	1
K	1	1	1	1	1	0

Figure 4: An example of access matrix

access restrictions cannot rely on the presence of a *reference monitor* (as in traditional systems) at the external server since it is not trusted to enforce the policy itself. Also, access control enforcement cannot be delegated to the data owner, since this practice presents the crucial shortcoming that it requires the data owner to be always involved in the processing of every access request. To overcome this issue, current proposals adopt a *multi-key encryption* schema where different data are encrypted with different keys. Although this multi-key approach is not new [32], the problem related to the definition, management, and evolution of the access control policy, and therefore of the corresponding encryption, introduces new challenges. We now discuss the proposals adopting a multi-key encryption schema for enforcing access control on outsourced data [16, 17].

The data owner defines an access control policy stating who can read what resources (write operations are performed at the owner's site). Notations $\mathcal{U}$ and $\mathcal{R}$ are used to denote the set of users and resources, respectively, in the system. Resources can be defined at different granularity levels (e.g., a resource may be a relation, a tuple, or even a cell). The access control policy is represented through an access matrix $\mathcal{A}$, with a row for each user in $\mathcal{U}$ and a column for each resource in $\mathcal{R}$. Each cell $\mathcal{A}[u_i,r_j]$ can assume two values: 1, if u_i is allowed to access r_j; 0, otherwise. Given an access matrix $\mathcal{A}$ over a set $\mathcal{U}$ of users and a set $\mathcal{R}$ of resources, $acl(r_j)$ denotes the access control list of resource r_j (i.e., the set of users who can access r_j).

Example 2.2. *Consider relation* PATIENTS *in Figure 1(a) and a set* $\mathcal{U}$=$\{$*Gilda* (*G*)*, Heidi* (*H*)*, Iris* (*I*)*, Jessica* (*J*)*, Kate* (*K*)$\}$ *of users. Figure 4 represents an example of an access matrix, where, for example,* $acl(t_1)$=$\{G,H,I,K\}$.

A naive solution for enforcing access control through encryption consists in encrypting each resource with a different key and communicating to each user the set of keys used to encrypt the resources she can access. Such a solution is clearly unacceptable, since each user has to manage as many keys as the number of resources she is authorized to access. To limit the number of keys that each user needs to store and manage a *key derivation* method is adopted. Basically, a key derivation method permits to compute an encryption key k_i starting from the knowledge of another key k_j and a piece of publicly available information. Key derivation methods are based on the definition of a *key derivation hierarchy* that specifies which keys can be derived from other keys in the system. A key derivation hierarchy can be graphically represented through a graph, with a vertex for each key and an edge from k_i to k_j iff k_j can be directly derived from k_i. Since key derivation can be recursively applied, a path in the graph from k_i to k_j represents the fact that key k_j can be directly or indirectly derived from k_i. The key derivation methods proposed in the literature can be classified depending on the supported key derivation hierarchy, which can be as follows.

- *Chain of vertices* (e.g., [41]): the key k_j of a vertex is computed by applying a one-way function to the key k_i of its predecessor in the chain and no public information is needed.

- *Tree hierarchy* (e.g., [42]): the key k_j of a vertex is computed by applying a one-way function to the key k_i of its direct ancestor in the tree and a publicly available label l_j associated with k_j.

- *DAG hierarchy* (e.g., [3, 4, 5, 14, 18]): a key k_j may have more than one direct ancestor and therefore its derivation is typically based on techniques that are more complex than the techniques used for chains and trees. Recent approaches [4, 5] working on DAGs are based on the definition of a set of *public tokens*. Given two keys k_i and k_j, a token $t_{i,j}$ is defined as $t_{i,j}{=}k_j{\oplus}f(k_i,l_j)$, where l_j is a publicly available label associated with k_j, $\oplus$ is the bitwise XOR operator, and f is a deterministic cryptographic function. Graphically, each token $t_{i,j}$ represents an edge in the key derivation hierarchy, connecting vertex k_i to

vertex k_j. Token $t_{i,j}$ allows the computation of k_j through k_i and l_j. The existence of a public token $t_{i,j}$ allows a user knowing k_i to derive key k_j through token $t_{i,j}$ and public label l_j.

Among the key derivation methods proposed in the literature, the technique in [4, 5] seems the solution that better fits the outsourced scenario since it minimizes the need of re-encrypting resources and generating new keys when the access control policy changes.

The access control policy defined by the data owner is then translated into an *equivalent encryption policy* $\mathcal{E}$, determining which data are encrypted with which key, the keys released to users, and the key derivation hierarchy. An encryption policy $\mathcal{E}$ is *equivalent* to an access control policy $\mathcal{A}$ if each user can decrypt all and only the resources she is authorized to access. To define such an encryption policy, the idea is to exploit a key derivation hierarchy induced by the set containment relationship $\subseteq$ over $\mathcal{U}$. This hierarchy has a vertex for each subset U of users in $\mathcal{U}$ and a path from vertex v_i to vertex v_j if v_i represents a subset of the users represented by v_j. The access control policy represented in $\mathcal{A}$ can be enforced by: *i)* assigning to each user the key associated with the vertex representing the user in the hierarchy; and *ii)* encrypting each resource with the key of the vertex corresponding to its access control list. In this way, a key can possibly be used to encrypt more than one resource, since all the resources with the same access control list are encrypted using the same key. Also, each user in the system has to manage only one key. It is easy to see that the encryption policy defined as mentioned above is equivalent to the access control policy defined by the data owner, since each user can derive only the keys associated with vertices that represent sets of users to which the user belongs. As a consequence, the user can only derive the keys used for encrypting resources for which she possesses the access privilege [17].

Example 2.3. *Consider the portion of the access matrix in Figure 4 that is defined on the set* $\mathcal{U}'=\{G,H,I,J\}$ *of users. Figure 5 illustrates the key derivation hierarchy defined over* $\mathcal{U}'$. *For readability, each vertex in the graph is associated with the set of users it represents. Here, dotted edges represent the associations user-key and resource-key. The encryption policy represented by*

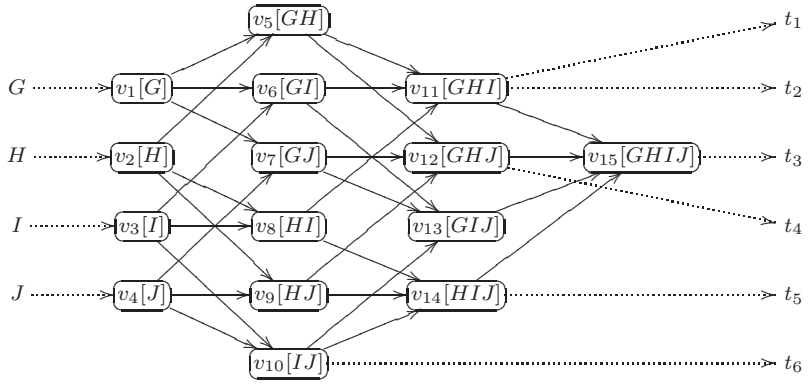


Figure 5: An example of encryption policy with a key hierarchy over $\mathcal{U}'$={G,H,I,J}

this key derivation hierarchy is equivalent to the access control policy in Figure 4. User Gilda, for example, knows key k_1 and therefore she can derive all and only the keys k_i associated with vertices in the hierarchy representing a set U of users including G (i.e., v_1, v_5-v_7, v_{11}-v_{13}, v_{15}). She can therefore decrypt tuples t_1, t_2, t_3, and t_4, as defined by the access matrix in Figure 4.

Although this solution is simple and easy to implement, it defines more keys than actually needed and requires the publication of a great number of tokens, which in turn makes key derivation less efficient. In fact, the tokens are stored in a public catalog on the server to make it available to any user. A key derivation then requires a series of client-server interactions. Since the problem of minimizing the number of tokens in the encryption policy $\mathcal{E}$, while guaranteeing equivalence with the access control policy $\mathcal{A}$ is NP-hard (it can be reduced to the set cover problem), in [17] the authors propose a heuristic algorithm working as follows.

- *Initialization.* The algorithm identifies the vertices necessary to enforce $\mathcal{A}$, called *material* vertices. Material vertices represent: *i)* singleton sets of users, whose keys are communicated to the users and that allow them to derive the keys of the resources they are entitled to access; and *ii)* the *acl*s of the resources, whose keys are used for encryption.

- *Covering.* For each material vertex v corresponding to a non-singleton set of users, the algorithm finds a set of material vertices that form a *non-redundant set covering* for v, which become direct ancestors of v. A set V of vertices is a set covering for v if for each

u in v, there is at least a vertex v_i in V such that u appears in v_i. It is non redundant if the removal of any vertex from V produces a set that does not cover v.

- *Factorization.* Whenever there is a set $\{v_1, \ldots v_m\}$ of vertices that have $n > 2$ common ancestors $v'_1, \ldots, v'_n$, it is convenient to insert an intermediate vertex v representing all the users in $v'_1, \ldots, v'_n$ and to connect each v'_i, $i = 1, \ldots, n$, with v, and v with each v_j, $j = 1, \ldots, m$. In this way, the encryption policy includes $n + m$, instead of $n \cdot m$ tokens in the catalog.

Example 2.4. *Consider the access control policy in Figure 4. During the initialization phase, the algorithm identifies the material vertices represented in Figure 6(a). Vertices $v_1, \ldots, v_5$ represent the encryption keys communicated to users, and vertices $v_6, \ldots, v_{10}$ represent the encryption keys used to protect resources. Figure 6(b) illustrates the key derivation hierarchy resulting from the covering phase of the algorithm, which correctly enforces the access control policy in Figure 4. It is easy to see that this hierarchy does not contain redundant edges. Figure 6(c) represents the key derivation hierarchy resulting from the factorization of vertices v_7 and v_8 that have three common direct ancestors (i.e., v_1, v_2, and v_5). To this purpose, the algorithm inserts non material vertex v_{11}, representing the set GHK of users, in the key derivation hierarchy. It then removes the 6 edges connecting v_1, v_2, and v_5 to v_7 and v_8, and inserts 3 edges connecting v_1, v_2, and v_5 to v_{11} and 2 edges connecting v_{11} to v_7 and v_8. We note that the hierarchy in Figure 6(b) has 15 edges, while the hierarchy in Figure 6(c) has 14 edges, thus saving one token.*

Whenever there is a change in the access control policy, the key derivation hierarchy and the resources involved in the change need to be appropriately updated. In particular, if the set of users who can access a resource r changes due to a grant or revoke operation, the data owner has to: download the encrypted version of r from the server; decrypt it; update the key derivation hierarchy if there is not a vertex representing the new set of users in $acl(r)$; encrypt the resource with the key k' associated with the vertex representing $acl(r)$; upload the new encrypted version

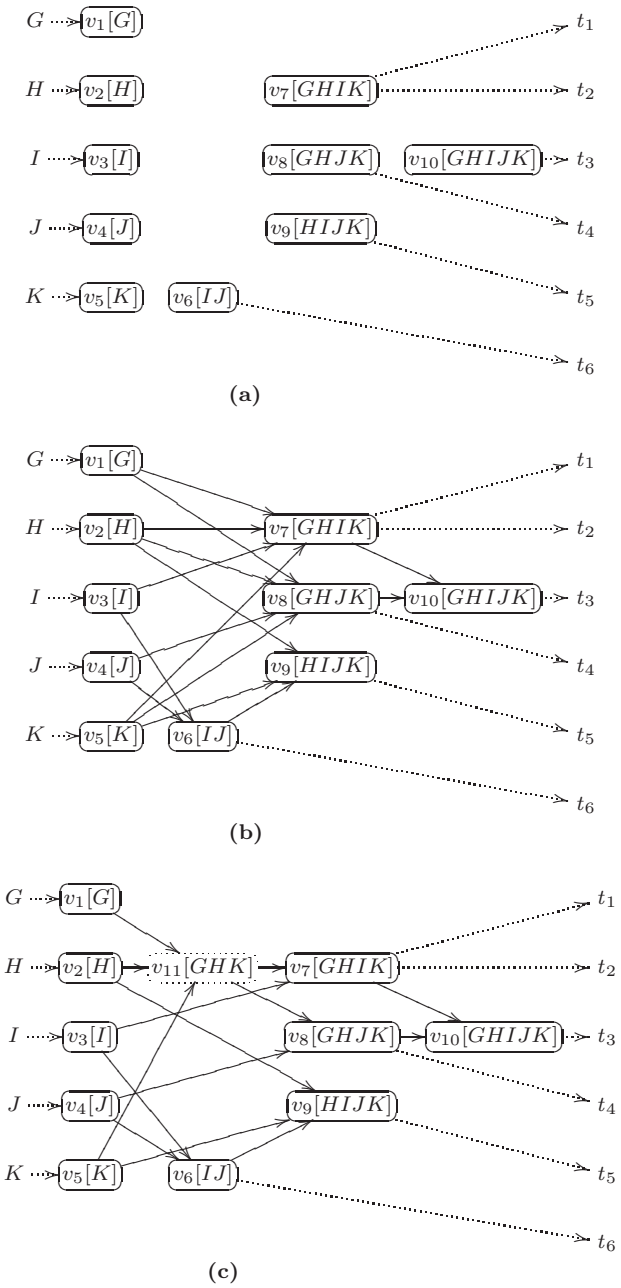


Figure 6: An example of initialization (a), covering (b), and factorization (c) generating an encryption policy equivalent to the access control policy in Figure 4

of r on the server; and possibly update the public catalog containing the tokens. To limit the burden for the data owner for managing updates to the access control policy, in [16] the authors propose a solution based on two layers of encryption, which partially delegates to the server the management of policy update operations. The first layer of encryption, called *Base Encryption*

Layer (BEL), is directly managed by the data owner, and enforces encryption on the resources according to the policy existing at initialization time. The BEL is only updated by possibly inserting tokens (i.e., edges in the key derivation hierarchy). The second layer of encryption, called *Surface Encryption Layer* (SEL), is managed by the server under the supervision of the data owner. The SEL is applied to encrypted resources, and enforces dynamic changes to the policy by possibly re-encrypting resources and changing the SEL key derivation hierarchy to correctly enforce updates of the access control policy. A user can then access a resource only if she knows the keys used at the SEL and at the BEL to encrypt the resource. Note that, since the server is in charge of managing the encryption of resources at the SEL, it may collude with a user of the system to decrypt resources that neither the server nor the user can access. This risk is limited, well defined, and can be reduced at the price of using a higher number of keys at the BEL, to minimize the situations where, due to a grant operation, resources encrypted with the same key at the BEL are encrypted with different keys at the SEL [17].

3 Fragmentation for protecting data confidentiality

The approaches described in Section 2 are based on the assumption that all data are sensitive and therefore they are protected by applying a layer of encryption before outsourcing. Encrypting the whole dataset has however the disadvantage that it is not always possible to efficiently execute queries and evaluate conditions over the encrypted data. Furthermore, often what is sensitive is the *association* among data more than the data per se. Recent approaches have therefore put forward the idea of using fragmentation for protecting data confidentiality and limiting the use of encryption [1, 9, 12, 13]. In the following, we first describe the basic concepts common to all the fragmentation-based proposals, and then present these proposals more in details.

3.1 Modeling confidentiality requirements

Given a relation r over relational schema $R(A_1, \ldots, A_n)$, with A_i an attribute on domain D_i, $i = 1, \ldots, n$, confidentiality requirements are modeled through *confidentiality constraints* [1, 9, 12, 13]. A confidentiality constraint c over relation schema $R(A_1, \ldots, A_n)$ is a subset of attributes in R (i.e., $c \subseteq R$). Confidentiality constraint c states that, for each tuple $t \in r$, the (joint) visibility of the values of the attributes in c is sensitive and must be protected. While simple, the definition of confidentiality constraints captures different protection requirements. Confidentiality constraints can be classified as *singleton* or *associations*. A singleton constraint states that the *values* of the attribute involved in the constraint are sensitive and cannot be released. For instance, the `SSN` of patients of a given hospital must be protected from disclosure. An association constraint states that the *association* among the values of the attributes in the constraint is sensitive and cannot be released. For instance, the association between the `Name` and the `Diagnosis` of a patient has to be protected from disclosure. The definition of confidentiality constraints is a complex problem that should take into consideration possible relationships among data. In this chapter, we assume that the data owner correctly defines confidentiality constraints on her data.

A set $\mathcal{C}$ of confidentiality constraints is *well defined* if it is non redundant, that is, iff $\forall c_i, c_j \in \mathcal{C}$, $i \neq j$, $c_i \not\subset c_j$. Intuitively, a constraint c_i such that $c_i \subseteq c_j$, with $i \neq j$, is redundant since the satisfaction of c_j implies the satisfaction of c_i. Note that any subset of attributes in c_j that does not represent a confidentiality constraint can be released.

Example 3.1. *Consider relation* Patients *in Figure 1(a), reported for simplicity in Figure 7(a). Figure 7(b) illustrates a set of well defined confidentiality constraints, modeling the following confidentiality requirements:*

- *the list of* `SSN`*s of patients is considered sensitive* (c_0)*;*
- *the association of patients' names with any other information in the relation is considered sensitive* ($c_1, \ldots, c_4$)*;*

PATIENTS

SSN	Name	DoB	County	Diagnosis	Prescription
782619730	Anne	55/01/23	Alameda	HIV	Nevirapine
946294626	Beth	86/04/05	Fresno	Anemia	Folic acid
737260262	Cheryl	40/12/23	Napa	Arthritis	Anti-inflammatory
937360965	Doris	81/07/22	Napa	Diabetes	Insulin
946259572	Evelyn	65/10/03	Orange	HeartAttack	Anticoagulants
837350362	Flora	89/03/24	Trinity	Diabetes	Insulin

(a)

c_0={SSN}
c_1={Name,DoB}
c_2={Name,County}
c_3={Name,Diagnosis}
c_4={Name,Prescription}
c_5={DoB,County,Diagnosis}
c_6={DoB,County,Prescription}

(b)

Figure 7: An example of relation (a) and of a set of well defined constraints over it (b)

- *attributes* DoB *and* County *can work as a quasi-identifier [39] and therefore can be exploited to infer the identity of patients; as a consequence, their associations with both* Diagnosis *and* Prescription *are considered sensitive* (c_5 *and* c_6).

Given a relation r defined over relation schema $R(A_1, \ldots, A_n)$, and a set $\mathcal{C}$ of confidentiality constraints, the data owner has to outsource r in a way that all sensitive attributes and associations modeled by $\mathcal{C}$ are properly protected. The approaches proposed in the literature address this problem by applying *fragmentation*, possibly combined with *encoding* techniques.

- *Fragmentation* consists in partitioning the attributes in relational schema R in different subsets (called fragments), which are then possibly outsourced in place of r. Formally, a *fragment* F_i is a subset of the attributes in R (i.e., $F_i \subseteq R$), and a *fragmentation* $\mathcal{F}$ is a set of fragments (i.e., $\mathcal{F} = \{F_1, \ldots, F_m\}$). Intuitively, fragmentation can be used to protect sensitive associations, by storing the attributes composing c in different fragments.

- *Encoding* consists in obfuscating an attribute (set thereof) such that its values are intelligible only to authorized users. Different solutions can be used to obfuscate attribute values [1] (e.g., encryption). Intuitively, encoding can be used to protect both sensitive values, by obfuscating them, or sensitive associations, by obfuscating at least one of the attributes involved.

Current solutions differ in how they fragment the original relation and if and how they adopt encoding to protect data confidentiality. In the remainder of this section, we will illustrate these

solutions in more details, providing a description of how data are fragmented and how queries can be executed on fragmented data.

3.2 Non-communicating servers

The first approach proposing the use of fragmentation combined with an encoding technique to protect the confidentiality of sensitive information has been presented in [1].

3.2.1 Data organization

Confidentiality constraints are enforced by fragmenting the original relation R into two fragments F_1 and F_2, stored at two non-communicating servers, who do not know each other. These fragments are obtained by partitioning the attributes in R in such a way that the attributes composing confidentiality constraints are not stored in the same fragment. Whenever one of the attributes in R cannot be stored within one of the two fragments without violating a confidentiality constraint, the attribute is encoded. The encoding of an attribute $A \in R$ consists in representing the values of the attribute with two different attributes A^1 and A^2 in F_1 and F_2, respectively. The original values of attribute A can be reconstructed only by authorized users, combining the values of attributes A^1 and A^2. For instance, A^1 may contain a random value *rnd*, and A^2 the result of the XOR between the original value of attribute A and *rnd* (i.e., $t[A] \oplus rnd$). Only a user knowing both A^1 and A^2 can reconstruct the original values of attribute A in R. A fragmentation $\mathcal{F}$ is then defined as a triple $\langle F_1, F_2, E \rangle$, where E is the set of encoded attributes that are stored plaintext neither at F_1 nor at F_2. A fragmentation $\mathcal{F} = \langle F_1, F_2, E \rangle$ is *correct* with respect to a set $\mathcal{C}$ of confidentiality constraints if $\forall c \in \mathcal{C}$, $c \not\subseteq F_1$ and $c \not\subseteq F_2$. Adopting this fragmentation model, singleton constraints can only be enforced by encoding the corresponding attribute. Association constraints can instead be satisfied by fragmentation, splitting the attributes in the constraints between F_1 and F_2. However, it is not always possible to satisfy an association constraint via fragmentation since it may happen that the attributes involved in an association constraint cannot be split between the two fragments

F_1^e

tid	SSN[1]	Name[1]	DoB	Diagnosis	Prescription
1	α	η	55/01/23	HIV	Nevirapine
2	β	Θ	86/04/05	Anemia	Folic acid
3	γ	ι	40/12/23	Arthritis	Anti-inflammatory
4	δ	κ	81/07/22	Diabetes	Insulin
5	ϵ	Λ	65/10/03	HeartAttack	Anticoagulants
6	ζ	μ	89/03/24	Diabetes	Insulin

F_2^e

tid	SSN[2]	Name[2]	County
1	τ	Ω	Alameda
2	υ	θ	Fresno
3	ϕ	σ	Napa
4	χ	λ	Napa
5	ς	ξ	Orange
6	ω	ς	Trinity

Figure 8: An example of correct fragmentation in the non-communicating servers scenario

without violating another constraint. In this case, an attribute in the constraint needs to be encrypted. We note that, besides correctness, $\mathcal{F}$ needs also to be *complete*, meaning that all the attributes in R are stored either plaintext or encoded in $\mathcal{F}$. Formally, a fragmentation $\mathcal{F}$=$\langle F_1,F_2,E\rangle$ is complete with respect to R if R=$F_1 \cup F_2 \cup E$.

At the physical level, fragmentation $\mathcal{F}$=$\langle F_1,F_2,E\rangle$ with $F_1 = \{A_{1_1}, \ldots, A_{1_i}\}$, $F_2 = \{A_{2_1}, \ldots, A_{2_j}\}$, and $E = \{A_{e_1}, \ldots, A_{e_l}\}$ translates into two *physical fragments* $F_1^e = \{\underline{\texttt{tid}}, A_{1_1}, \ldots, A_{1_i}, A_{e_1}^1, \ldots, A_{e_l}^1, \}$ and $F_2^e = \{\underline{\texttt{tid}}, A_{2_1}, \ldots, A_{2_j}, A_{e_1}^2, \ldots, A_{e_l}^2\}$. Attribute `tid` is the primary key of both physical fragments and guarantees the *lossless join* property. It can be either: *1)* the key attribute of the original relation R, if it does not violate any confidentiality constraint when stored in F_1 or in F_2, or *2)* an attribute that is added to both F_1^e and F_2^e. The presence of this attribute guarantees the possibility, for authorized users, to reconstruct the content of the original relation R.

Example 3.2. *Consider relation* PATIENTS *in Figure 7(a) and the set of confidentiality constraints over it in Figure 7(b). An example of correct and complete fragmentation is* $\mathcal{F}$=$\{\{$`DoB`, `Diagnosis`, `Prescription`$\}$, {`County`}, {`SSN`, `Name`}$\}$. *Figure 8 illustrates the corresponding physical fragments, where, for simplicity, obfuscated values are represented with Greek letters. Constraint* c_0 *is singleton and therefore can be satisfied only by obfuscating attribute* `SSN`. *Association constraints* $c_1, \ldots, c_4$ *are satisfied by obfuscating attribute* `Name`, *and constraints* c_5 *and* c_6 *are solved via fragmentation. We note that attribute* `Name` *cannot be stored in the clear in a fragment without violating at least a constraint.*

In general, given a relation R and a set of well defined constraints $\mathcal{C}$ over it, different correct and complete fragmentations may exist. As an example, the fragmentation $\mathcal{F}$=$\langle F_1,F_2,E\rangle$, with E=R, that encodes all the attributes in R is always correct and complete. However, such a solution is equivalent to encrypt the whole relation, thus nullifying the advantages that the availability of data in clear form may have on query execution. Among all the fragmentations of R that enforce $\mathcal{C}$, the one minimizing query evaluation costs at the client side is then to be preferred. To evaluate the query execution cost, in [1] the authors adopt an *affinity matrix* M, with a row and a column for each attribute in R. Each entry $M[A_i, A_j]$ represents the cost that would be paid in query evaluation if attributes A_i and A_j $(i \neq j)$ are not stored in the same fragment. Each entry $M[A_i, A_i]$ represents the cost that would be paid if attribute A_i is encoded (i.e., $A_i \in E$). As a consequence, the cost of a fragmentation $\mathcal{F}$ is defined as the sum of the cells $M[A_i, A_j]$ such that $A_i \in F_1$ and $A_j \in F_2$, and of the cells $M[A_i, A_i]$ such that $A_i \in E$.

In [1] the authors show that the problem of computing a fragmentation with minimum cost is NP-hard (the minimum hypergraph coloring problem reduces to it). They therefore propose three different heuristics working in polynomial time in the number of attributes in R, which are obtained by combining known approximation algorithms for the min-cut and the weighted set cover problems.

3.2.2 Query execution

The query execution process must be revised to take into consideration the fact that relation R is stored in two physical fragments F_1^e and F_2^e managed by two external servers S_1 and S_2, respectively. Consider a query q of the form SELECT A FROM R WHERE $Cond$, where A is a subset of attributes in R, and $Cond = \bigwedge_i cond_i$ is a conjunction of basic predicates of the form $(A_i \ op \ v)$, $(A_i \ op \ A_j)$, or $(A_i$ IN $\{v_i, \ldots, v_k\})$, where A_i, $A_j \in R$, $\{v, v_i, \ldots, v_k\}$ are constant values in the domain of attribute A_i, and op is a comparison operator in $\{=, \neq, >, <, \geq, \leq\}$. Query q is first reformulated as a query operating on the join between the two fragments F_1^e and F_2^e. The query execution plan can then be determined by simply generalizing and applying

the standard database optimization techniques. This implies that projections can be pushed down to the servers, just taking care to not project out attribute `tid` necessary for the join between F_1^e and F_2^e. Selection conditions can be pushed down to the server storing the involved attributes in clear form. More precisely, the conditions specified in the WHERE clause are split into: $Cond_{s_1}$ that is the conjunction of all basic conditions that involve attributes stored in F_1^e only; $Cond_{s_2}$ that is the conjunction of all basic conditions that involve attributes stored in F_2^e only; and $Cond_c$ that is the conjunction of all basic conditions defined on encoded attributes or involving attributes that are not stored in the same fragment. Conditions $Cond_{s_1}$ and $Cond_{s_2}$ can then be pushed down to servers S_1 and S_2, respectively, for their evaluation, while $Cond_c$ can be executed only at the client side together with the join operation.

Once the logical plan has been optimized, it is necessary to determine the physical execution plan that establishes how the query execution is partitioned among the two servers and the client. To this purpose, the following two strategies can be applied.

- *Parallel strategy*. The two servers first evaluate conditions $Cond_{s_1}$ and $Cond_{s_2}$, and then the client computes the join between the results returned by the two external servers and evaluates $Cond_c$.

- *Sequential strategy*. One of the two external servers first evaluates condition $Cond_{s_i}$ and returns the result to the client who, in turn, sends the projection over attribute `tid` to the other storage server. The second server then evaluates $Cond_{s_j}$ on the subset of tuples indicated by the client. Finally, the client refines the result returned by S_j evaluating $Cond_c$.

The parallel strategy may be more expensive than the sequential strategy since much more data are transferred from the two servers to the client who has then to perform the join operation. The sequential strategy implies a sequential execution of the queries and may cause privacy breaches since if the server receiving the values of attribute `tid` knows query q, then it can infer what tuples satisfy the conditions evaluated on the other server.

Parallel strategy			Sequential strategy		
r_{s_1} :=	SELECT	tid, Name^1	r_{s_1} :=	SELECT	tid, Name^1
	FROM	F_1^e		FROM	F_1^e
	WHERE	Diagnosis='Diabetes'		WHERE	Diagnosis='Diabetes'
r_{s_2} :=	SELECT	tid, Name^2	r_{s_2} :=	SELECT	tid, Name^2
	FROM	F_2^e		FROM	F_2^e
	WHERE	County='Napa'		WHERE	County='Napa' AND tid IN {4,6}
r_c :=	SELECT	Name^1 $\oplus$ Name^2 AS Name	r_c :=	SELECT	Name^1 $\oplus$ Name^2 AS Name
	FROM	r_{s_1} JOIN r_{s_2} ON r_{s_1}.tid=r_{s_2}.tid		FROM	r_{s_1} JOIN r_{s_2} ON r_{s_1}.tid=r_{s_2}.tid

Figure 9: An example of query translation in the non-communicating servers scenario

Example 3.3. *Consider relation* PATIENTS *in Figure 7(a), the set of constraints over it in Figure 7(b), and the fragments in Figure 8. Suppose that a client formulates query* q = SELECT Name FROM PATIENTS WHERE Diagnosis=*'Diabetes'* AND County=*'Napa', returning the names of patients living in the Napa county and suffering from diabetes. The conditions operating at the different parties are:* $Cond_{s_1}$={Diagnosis=*'Diabetes'*}, $Cond_{s_2}$={County=*'Napa'*}, *and* $Cond_c$=$\emptyset$, *respectively. Figure 9 illustrates the translation of query* q *in the queries, for both the parallel and sequential strategies, operating at each server and at the client side. Here, we denote with* r_{s_1}, r_{s_2}, *and* r_c *the results of queries* q_{s_1}, q_{s_2}, *and* q_c, *respectively. The set* {*4,6*} *appearing in* q_{s_2} *of the sequential strategy represents the identifiers of the tuples in* r_{s_1}, *that is, the tuples with* Diagnosis=*'Diabetes'.*

3.3 Multiple fragments

The main limitation of the solution in [1] is that it relies on the absence of communication between the two storage servers. This assumption is clearly difficult to satisfy in a real world scenario where a collusion between the servers or the users accessing the data can cause privacy breaches. To overcome this problem, in [9] the authors present a solution where several fragments can be all stored at the same server.

3.3.1 Data organization

Confidentiality constraints are enforced by fragmenting the original relation R into a set of unlinkable fragments. The enforcement of association constraints via fragmentation is always possible since the number of fragments composing $\mathcal{F}$ is not fixed a priori. As a consequence, if an attribute cannot be inserted into an existing fragment without violating a confidentiality constraint, a new fragment is created and the attribute is inserted in it. All attributes that do not appear in a singleton constraint can then be represented in the clear in a fragment in $\mathcal{F}$, thus improving query execution efficiency. Furthermore, we say that $\mathcal{F}$ *maximizes visibility*, since encryption is only applied to protect the values of attributes appearing in singleton constraints. A fragmentation $\mathcal{F} = \{F_1, \ldots, F_n\}$ is *correct* with respect to a set $\mathcal{C}$ of confidentiality constraints if: *i)* $\forall c \in \mathcal{C}, \forall F \in \mathcal{F},\ c \not\subseteq F$, and *ii)* $\forall F_i, F_j \in \mathcal{F}, i \neq j,\ F_i \cap F_j = \emptyset$. The first condition states that a fragment cannot contain in the clear attributes that form a confidentiality constraint. The second condition states that the fragments must be disjoint. In this way, the fragments composing a fragmentation can be stored at the same external server, since they cannot be joined to reconstruct the content of the original relation.

At the physical level, each fragment $F_i{=}\{A_{i_1}, \ldots, A_{i_n}\}{\in}\mathcal{F}$ translates into a physical fragment $F_i^e(\underline{salt}, enc, A_{i_1}, \ldots, A_{i_n})$, where attribute *salt* is the primary key of F_i^e and contains a randomly chosen value, and attribute *enc* represents the encryption of the attributes in R that do not belong to the fragment (i.e., $R \setminus F_i$), combined before encryption in a binary XOR with the salt to prevent frequency attacks [43].

Example 3.4. *Consider relation* PATIENTS *in Figure 7(a) and the set of confidentiality constraints over it in Figure 7(b). An example of correct fragmentation that maximizes visibility is* $\mathcal{F}{=}\{\{$`Name`$\}, \{$`DoB`, `County`$\}, \{$`Diagnosis`, `Prescription`$\}\}$. *Figure 10 illustrates the physical fragments storing* $\mathcal{F}$. *Note that the unique attribute in relation* PATIENTS *that does not appear in the clear in any fragment is* `SSN`, *since it forms a singleton constraint* (c_0).

Given a relation R and a set of well defined constraints $\mathcal{C}$ over it, different correct fragmen-

F_1^e

salt	enc	Name
s_1^1	α	Anne
s_2^1	β	Beth
s_3^1	γ	Cheryl
s_4^1	δ	Doris
s_5^1	ε	Evelyn
s_6^1	ζ	Flora

F_2^e

salt	enc	DoB	County
s_1^2	ϑ	55/01/23	Alameda
s_2^2	ι	86/04/05	Fresno
s_3^2	κ	40/12/23	Napa
s_4^2	λ	81/07/22	Napa
s_5^2	μ	65/10/03	Orange
s_6^2	ν	89/03/24	Trinit

F_3^e

salt	enc	Diagnosis	Prescription
s_1^3	τ	HIV	Nevirapine
s_2^3	υ	Anemia	Folic acid
s_3^3	ϕ	Arthritis	Anti-inflammatory
s_4^3	φ	Diabetes	Insulin
s_5^3	χ	HeartAttack	Anticoagulants
s_6^3	ς	Diabetes	Insulin

Figure 10: An example of correct fragmentation in the multiple fragments scenario

tations that guarantee maximal visibility may exist. For instance, a fragmentation $\mathcal{F}$ where each attribute that does not appear in a singleton constraint is stored in a different fragment is correct and guarantees maximal visibility. Such a fragmentation however makes query execution inefficient. In fact, the evaluation of any query operating on more than one attribute always requires the client involvement. It is then essential to determine a correct fragmentation maximizing visibility and limiting the query evaluation burden for the client. The following metrics have been proposed to measure the quality of a fragmentation.

- *Number of fragments* [9]. A straightforward approach for computing a fragmentation that reduces the query evaluation costs consists in minimizing the number of fragments. The rationale is that with a low number of fragments, more attributes in clear are stored in the same fragment, thus allowing a more efficient query execution.

- *Affinity between attributes* [13]. The quality of a fragmentation can be measured in terms of the affinity between pairs of attributes, which is represented through a traditional affinity matrix. A fragmentation with high affinity is likely to reduce the query evaluation costs at the client side.

- *Query cost function* [11]. If the expected query workload for R is known, a specific cost function that models the cost of evaluating queries on $\mathcal{F}$ can be precisely defined and minimized.

The problem of computing a fragmentation that minimizes the workload of query execution at the client side with respect to the three metrics above is NP-hard (either the minimum

hypergraph coloring or the minimum hitting set problems reduce to it [9, 11, 13]). The heuristic algorithms proposed for solving such a problem are all based on the definition of a partial order relationship, called *dominance relationship* and denoted $\preceq$, on fragmentations. A fragmentation $\mathcal{F}'$ dominates a fragmentation $\mathcal{F}$, denoted $\mathcal{F}\preceq\mathcal{F}'$, if $\mathcal{F}'$ can be obtained by merging two or more fragments in $\mathcal{F}$.

Example 3.5. *Consider relation* PATIENTS *in Figure 7(a), the set of constraints over it in Figure 7(b), and the following two correct fragmentations that maximize visibility:* $\mathcal{F}_1$={{`Name`}, {`DoB`, `County`}, {`Diagnosis`, `Prescription`}} *and* $\mathcal{F}_2$={{`Name`}, {`DoB`}, {`County`}, {`Diagnosis`, `Prescription`}}. *Since* $\mathcal{F}_1$ *can be obtained by merging fragments* {`DoB`} *and* {`County`} *in* $\mathcal{F}_2$, $\mathcal{F}_2\preceq\mathcal{F}_1$.

Note that if $\mathcal{F}\preceq\mathcal{F}'$, then $\mathcal{F}'$ is clearly composed of a lower number of fragments than $\mathcal{F}$, its affinity is higher than the affinity of $\mathcal{F}$, and the evaluation of a query cost function over it results lower than the evaluation of the same function over $\mathcal{F}$. The rationale is that if the number of plaintext attributes in a fragment increases, then the affinity increases and the query costs decrease. We can conclude that, given two fragmentations $\mathcal{F}$ and $\mathcal{F}'$, with $\mathcal{F}\preceq\mathcal{F}'$, $\mathcal{F}'$ is always more convenient than $\mathcal{F}$, independently from the metric used to measure the quality of the solution. The heuristics proposed in [9, 11, 13] aim therefore at computing a *minimal fragmentation* that satisfies the following three conditions: *i)* $\mathcal{F}$ is correct; *ii)* $\mathcal{F}$ maximizes visibility (i.e., association constraints are enforced through fragmentation only); and *iii)* there is not another fragmentation $\mathcal{F}'$ that is correct, maximizes visibility, and that dominates $\mathcal{F}$ (i.e., $\mathcal{F}\preceq\mathcal{F}'$). The algorithms proposed in [9, 11, 13] work in polynomial time in the number of attributes composing R and constraints in $\mathcal{C}$.

3.3.2 Query execution

Like for the fragmentation model in [1], the query execution process must be revised. Since each physical fragment F^e represents, either plaintext or encrypted, all the attributes in R, any query q operating on R can be evaluated accessing one physical fragment only. Consider

a query q of the form SELECT A FROM R WHERE $Cond$, where A is a subset of attributes in R, and $Cond{=}\bigwedge_i cnd_i$ is a conjunction of basic predicates. The translation process consists in first splitting $Cond{=}\bigwedge_i cnd_i$ into two sets of conditions, depending on the attributes that each basic condition in $Cond$ involves and therefore on the party that can evaluate it. In particular, if F^e is the fragment chosen for query evaluation, $Cond$ is split into: $Cond_s$ that is the conjunction of basic conditions that involve only attributes plaintext represented in F^e; and $Cond_c$ that is the conjunction of basic conditions involving at least an attribute that is not plaintext represented in F^e. The external server then evaluates condition $Cond_s$ and returns the tuples satisfying it to the client. The client decrypts attribute *enc*, evaluates $Cond_c$, and projects the resulting relation over the set A of attributes. We note that the choice of the fragment on which query q must be evaluated should minimize the computational overhead at the client side, and therefore limit the number of tuples satisfying $Cond_s$ that are returned to the client. A possible strategy for choosing the fragment may consist in selecting the fragment on which it is possible to directly execute the most selective conditions, to reduce the amount of data returned to the client.

Example 3.6. *Consider relation* PATIENTS *in Figure 7(a), the set of constraints over it in Figure 7(b), and the fragmentation in Figure 10. Suppose now that a client formulates query q* = SELECT `Name` FROM PATIENTS WHERE `Diagnosis`='*Diabetes*' AND `County`='*Napa*' *returning the names of patients living in the Napa county and suffering from diabetes. The query can be translated to operate on each of the three fragments, but the evaluation using either* F_2^e *or* F_3^e *is more convenient than using* F_1^e*, since they contain a subset of the attributes appearing in the conditions of q. The translation of query q in the corresponding queries operating at the server and at the client side, using either* F_2^e *or* F_3^e*, are illustrated in Figure 11.*

3.4 Departing from encryption

Although the solution illustrated in Section 3.3 limits the adoption of encryption to the attributes that appear in singleton constraints, encryption carries the burden of key management

Translation over F_2^e			Translation over F_3^e		
$r_s :=$	SELECT	*salt*, *enc*	$r_s :=$	SELECT	*salt*, *enc*
	FROM	F_2^e		FROM	F_3^e
	WHERE	`County`='Napa'		WHERE	`Diagnosis`='Diabetes'
$r_c :=$	SELECT	`Name`	$r_c :=$	SELECT	`Name`
	FROM	*Decrypt*(r_s.*enc*,r_s.*salt*,*k*)		FROM	*Decrypt*(r_s.*enc*,r_s.*salt*,*k*)
	WHERE	`Diagnosis`='Diabetes'		WHERE	`County`='Napa'

Figure 11: An example of query translation in the multiple fragments scenario

and makes query execution expensive. In [10, 12] the authors propose an approach that completely departs from encryption. This proposal is based on the assumption that the data owner is willing to store a small portion of the data to satisfy confidentiality constraints.

3.4.1 Data organization

Confidentiality constraints are enforced by fragmenting the original relation R into two fragments F_o and F_s, where F_o is stored at the data owner and F_s is stored at the external server. Intuitively, fragment F_o contains sensitive attributes (singleton constraints) and at least one attribute for each sensitive association. In this way, sensitive attributes as well as sensitive associations are not exposed to the external server. A fragmentation $\mathcal{F}$=$\langle F_o,F_s\rangle$ is *correct* with respect to a set $\mathcal{C}$ of confidentiality constraints if $\forall c \in \mathcal{C}$, $c \not\subseteq F_s$. We note that fragment F_o does not need to satisfy this condition (i.e., F_o can possibly violate constraints), since it is stored at the owner side and only authorized users can access it. Furthermore, to avoid loss of information, all attributes in R should be represented either in F_o or in F_s (i.e., $F_o \cup F_s = R$) (*completeness* property). At the physical level, fragmentation $\mathcal{F}$=$\langle F_o,F_s\rangle$ with $F_o = \{A_{o_1}, \ldots, A_{o_i}\}$ and $F_s = \{A_{s_1}, \ldots, A_{s_j}\}$ translates into two physical fragments $F_o^e(\underline{\texttt{tid}},A_{o_1},\ldots,A_{o_i})$ and $F_s^e(\underline{\texttt{tid}},A_{s_1},\ldots,A_{s_j})$. Attribute `tid` is the primary key of both physical fragments and guarantees the *lossless join* property (i.e., the correct reconstruction of the original relation). It can be either: *1)* the key attribute of R, if it does not violate confidentiality constraints when added in F_s, or *2)* an attribute that is added to both F_o^e and F_s^e during the fragmentation process. Note that the attributes stored in fragment F_s should not be replicated

F_o^e

tid	SSN	Name	County
1	782619730	Anne	Alameda
2	946294626	Beth	Fresno
3	737260262	Cheryl	Napa
4	937360965	Doris	Napa
5	946259572	Evelyn	Orange
6	837350362	Flora	Trinity

F_s^e

tid	DoB	Diagnosis	Prescription
1	55/01/23	HIV	Nevirapine
2	86/04/05	Anemia	Folic acid
3	40/12/23	Arthritis	Anti-inflammatory
4	81/07/22	Diabetes	Insulin
5	65/10/03	HeartAttack	Anticoagulants
6	89/03/24	Diabetes	Insulin

Figure 12: An example of correct fragmentation departing from encryption

in F_o (*non-redundancy* property) to avoid unnecessary storage at the data owner side and usual replica management problems.

Example 3.7. *Consider relation* PATIENTS *in Figure 7(a) and the set of well defined constraints over it in Figure 7(b). An example of a correct fragmentation is* F_o=$\{$`SSN`, `Name`, `County`$\}$ *and* F_s=$\{$`DoB`, `Diagnosis`, `Prescription`$\}$. *Figure 12 illustrates the corresponding physical fragments, where an artificial tuple identifier has been added.*
Constraint c_0 *is satisfied by storing attribute* `SSN` *in* F_o. *Constraints* $c_1, \ldots, c_4$ *are satisfied by storing attribute* `Name` *in* F_o. *Constraints* c_5 *and* c_6 *are satisfied by storing attribute* `County` *in* F_o.

Given a relation R and a set of well defined confidentiality constraints $\mathcal{C}$ over it, there may exist different fragmentations that are correct, complete, and non-redundant. For instance, a fragmentation $\mathcal{F}$=$\langle F_o, F_s \rangle$, with F_o=R and F_s=$\emptyset$, is correct, complete, and non-redundant but it is clearly not acceptable since it corresponds to not outsourcing the data. It is then needed to compute a fragmentation that minimizes the data owner's workload in terms either of storage, computation, or both. In [12] the authors illustrate the following metrics to measure the quality of a fragmentation, which differ in the resource whose consumption should be minimized (storage vs computation) and on the information available about the system workload at fragmentation time.

- *Number of attributes*: the cost of a fragmentation corresponds to the number of attributes in F_o.

- *Storage space*: the cost of a fragmentation corresponds to the physical size of the attributes in F_o.

- *Number of queries*: the cost of a fragmentation corresponds to the number of queries that involve at least one of the attributes in F_o.

- *Number of conditions*: the cost of a fragmentation corresponds to the number of conditions in queries that involve at least one of the attributes in F_o.

The problem of computing a fragmentation that minimizes one of the metrics above-mentioned is NP-hard (the minimum hitting set problem reduces to it [12]). A heuristic algorithm has been proposed that solves the problem for any metric and works in polynomial time with respect to the number of attributes in R.

3.4.2 Query execution

Similarly to the non-communicating servers approach, a query formulated by users on R must be translated into queries operating on F_o^e and F_s^e. Consider a query q of the form SELECT A FROM R WHERE $Cond$, where A is a subset of attributes in R, and $Cond = \bigwedge_i cond_i$ is a conjunction of basic predicates. As for the fragmentation techniques previously described, condition $Cond$ is first split into: $Cond_o$ that is the conjunction of basic conditions operating only on attributes in F_o; $Cond_s$ that is the conjunction of basic conditions involving only attributes in F_s; and $Cond_{so}$ that is the conjunction of basic conditions involving both attributes in F_o and F_s. Condition $Cond_s$ can be pushed down to the server, while conditions $Cond_o$ and $Cond_{so}$ are executed at the owner side, possibly with the support of the server.

The evaluation of a query q can proceed according to the following two different strategies, depending on the order in which conditions $Cond_s$, $Cond_o$, and $Cond_{so}$ are evaluated.

- *Server-Owner strategy*. The external server evaluates condition $Cond_s$, the data owner then computes the join between the result returned by the external server and F_o and evaluates $Cond_o$ and $Cond_{so}$.

Server-Owner strategy

$r_s :=$ SELECT `tid`
FROM F_s^e
WHERE `Diagnosis`='Diabetes'

$r_{so} :=$ SELECT `Name`
FROM F_o^e JOIN r_s ON F_o^e.`tid`=r_s.`tid`
WHERE `County`='Napa'

Owner-Server strategy

$r_o :=$ SELECT `tid`
FROM F_o^e
WHERE `County`='Napa'

$r_s :=$ SELECT `tid`
FROM F_s^e
WHERE `Diagnosis`='Diabetes' AND `tid` IN {3,4}

$r_{so} :=$ SELECT `Name`
FROM F_o^e JOIN r_s ON F_o^e.`tid`=r_s.`tid`

Figure 13: An example of query translation departing from encryption

- *Owner-Server strategy*. The data owner evaluates condition $Cond_o$ and sends the projection over attribute `tid` to the external server. The external server evaluates condition $Cond_s$ on the subset of tuples indicated by the data owner. Finally, the data owner refines the result received from the server, by evaluating condition $Cond_{so}$.

The choice between these two strategies should take into consideration, besides efficiency in query evaluation, the fact that if query q is publicly available, the Owner-Server strategy reveals to the server the tuples that satisfy condition $Cond_o$, thus possibly causing privacy breaches.

Example 3.8. *Consider relation* PATIENTS *in Figure 7(a), the set of constraints over it in Figure 7(b), and the fragmentation in Figure 12. Suppose now that a client formulates query* $q =$ SELECT `Name` FROM PATIENTS WHERE `Diagnosis`='Diabetes' AND `County`='Napa' *returning the names of patients living in the Napa county and suffering from diabetes. The conditions operating at the different parties are:* $Cond_s$={`Diagnosis`='Diabetes'}, $Cond_o$={`County`='Napa'}, *and* $Cond_{so}$=$\emptyset$, *respectively. The translation of* q *into the corresponding queries, for the Server-Owner and Owner-Server strategies are illustrated in Figure 13. Set* {3,4} *used in query* q_s *of the Owner-Server strategy represents the identifiers of the tuples in* r_o, *that is, of the tuples with* `County`='Napa'.

4 Protecting data integrity

Besides protecting data confidentiality, it is also necessary to design mechanisms for protecting the integrity and authenticity of the data. As a matter of fact, users as well as organizations are increasing their dependency on data for their daily operations, thus making data integrity a critical issue. Guaranteeing integrity means that techniques should be adopted to easily verify that the external server does not improperly modify data in storage and that the server provides a correct response to queries (i.e., the server does not delete or modify data improperly). In this section, we illustrate the main techniques proposed to guarantee data integrity in storage and query computation.

4.1 Integrity in storage

Data integrity can be provided at different granularity levels: table, attribute, tuple, or cell level. The verification of the integrity at the table level and attribute level can be performed by the client only if she receives the whole table/column. Data integrity at the cell level suffers from a high verification overhead. For these reasons, the majority of the proposals in the literature provide data integrity at the tuple level and rely on *digital signatures* (e.g., [25]). The data owner first signs, with her private key, each tuple t in a relation, and the signature is concatenated to the corresponding tuple. The relation is then encrypted and outsourced to the external server. When a client receives a set of tuples from the external server, she can check the signature associated with the tuples to detect possible unauthorized changes to the data. The main drawback of this solution is that the verification cost at the client side grows linearly with the number of tuples in the query result. To limit this burden, in [34] the authors propose the adoption of a schema that permits to combine the signature of a set of tuples in a unique signature. To this purpose, the authors consider three different signature schemas: *condensed RSA* encryption schema, a variation of traditional RSA encryption schema, which allows the aggregation of signatures generated by the same signer; *BGLS* encryption schema [6] based on

bilinear mappings, which supports the aggregation of signatures generated by different signers; batch *DSA signature aggregation* whose verification is based on the multiplicative homomorphic property of these signatures. The signature verification processes for the condensed RSA and BGLS schemas are more efficient than the signature verification for the batch DSA schema. Both condensed RSA and BGLS approaches are *mutable*, meaning that any user who knows multiple aggregated signatures can compose them, obtaining a valid aggregated signature that may correspond to the aggregate signature of an arbitrary set of tuples. Although this feature can be of interest in the process of generating aggregated signatures, it also represents a weakness for the integrity of the data. In [33], the authors propose an extension to condensed RSA and BGLS techniques that makes them immutable. Such an extension is based on zero knowledge protocols that allow a server to only reveal a proof of the knowledge of the aggregated signature associated with a query result, instead of revealing the signature itself.

4.2 Integrity in query computation

In addition to provide assurance on the fact that data stored at external servers are protected from unauthorized changes (data integrity in storage), it is also becoming more and more important to guarantee the correctness and completeness of query results. The verification of the integrity of data processing results is particularly difficult to implement, especially in the emerging large-scale platforms used, for example, in cloud computing. The approaches proposed in the literature can be classified in the following two categories.

- *Authenticated data structures* approaches (e.g., [19, 29, 31, 35, 36, 37, 50]) are based on the definition of an appropriate data structure (e.g., a signature chaining, a Merkle hash tree, or a skip list). These solutions provide completeness guarantee for the queries operating on the attribute (set thereof) on which the data structure has been defined. Note that all these approaches also guarantee data integrity in storage since unauthorized changes to the data can be detected during the integrity verification process of query results.

- *Probabilistic* approaches (e.g., [30, 46, 48]) are based on the insertion of sentinels in the outsourced data, which must also belong to the query result. These solutions provide a probabilistic guarantee of completeness of query results.

Authenticated data structures approaches. One of the first solutions is the *signature chaining* approach [35, 36], which has been proposed to verify the completeness of the result of range queries. Given an attribute A defined over domain D and characterized by a total order relationship, the content of the outsourced relation is ordered with respect to the value that attribute A assumes in each tuple. The signature associated with each tuple t_i is then computed by signing the string resulting from the concatenation of $h(t_{i-1})$ with $h(t_i)$, where h is a one-way hash function and t_{i-1} is the tuple preceding t_i in the order defined by attribute A over the outsourced relation. If the result of a range query operating on attribute A is not complete since one tuple, say t_i, has been omitted by the external server, the signature verification process reveals that the result is not complete. In fact, during the verification process, the client computes a signature for tuple t_{i+1}, that is $h(t_{i-1})||h(t_{i+1})$, which is different from the original signature associated with t_{i+1} (i.e., $h(t_i)||h(t_{i+1})$). The main limitation of this solution is that it guarantees the completeness of the query result only with respect to the attribute on which the signature chain has been defined. A signature chain has to be defined for each attribute that may be involved in a range query. As a consequence, the size of the signature associated with each tuple, and therefore also the time necessary for its verification, increases linearly with the number of signature chains.

Other approaches are based on the definition of a *Merkle hash tree* [31]. A Merkle hash tree is a binary tree, where the leaves contain the hash of one tuple of the outsourced relation, and each internal node contains the result of the application of a one-way hash function on the concatenation of the children of the node itself. The root of the Merkle hash tree is signed by the data owner and communicated to authorized users. The tuples in the leaves of the tree are ordered according to the value of a given attribute A. Figure 14 illustrates an example of a Merkle hash tree built over relation PATIENTS in Figure 1(a) for attribute `Name`. Whenever

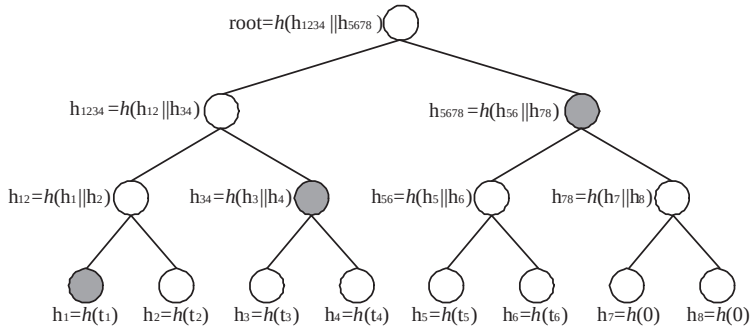


Figure 14: An example of Merkle hash tree

the external server evaluates a range query operating on A, it returns to the requesting client the result of the query along with a *verification object* (VO) including all the information that the client needs to know to verify the completeness of the query result [19]. In particular, the computation of the verification object depends on the type of query submitted to the server. For instance, in case of a selection query that returns a specific tuple, the verification object includes the values of all the nodes that are sibling of the nodes in the path from the leaf corresponding to the returned tuple to the root of the tree. The content of the verification object is necessary to recompute the value of the root and to verify whether the computed value is equal to the value originally signed by the data owner. If the comparison succeeds, the query result is correct; it is not correct, otherwise. For instance, with respect to the Merkle hash tree in Figure 14, the verification object for a query that returns the patient whose name is Beth (i.e., tuple t_2) is represented by gray nodes in the figure. After the proposal in [19], many different solutions have been presented with the goal of improving the efficiency of the verification process [29, 37] and to support join operations [50]. Like for signature chaining, also the adoption of Merkle hash trees requires the definition of a different data structure for each attribute that can be involved in a query.

Since Merkle hash trees cannot efficiently support updates to the outsourced data, in [20] the authors propose to use an *authenticated skip list* for verifying the completeness of query results. A skip list is a hierarchical data structure that stores an ordered list of elements and efficiently supports the search, insertion, and removal of elements in the list. The proposal

in [20] consists in representing through a relational table, called *security table*, the skip list built over the outsourced relation R. To prove the completeness of a query, the user checks the value of few tuples in the security table. In [20], the authors describe different techniques for efficiently querying the security table and illustrate how skip lists easily support updates to the outsourced relation.

Probabilistic approaches. The solutions based on the definition of an authenticated data structure have the advantage that they provide a guarantee of the completeness of query results with absolute certainty. The main problem however is that these data structures can be used only for the specific attribute on which they are built. This implies that the completeness of queries operating on different attributes cannot be checked. To solve this problem, probabilistic approaches allow a client to verity the completeness of any query result with high probability, reducing also the performance overhead. In [48] the authors propose a probabilistic approach based on the insertion of *fake tuples* in the relations before outsourcing them. When a client receives the query result, it checks whether the fake tuples that satisfy the conditions specified in the query belong to the result. If at least a fake tuple is missing, the query result is not complete. Clearly, fake tuples must be indistinguishable at the server's eye from real tuples, since otherwise the server could compute the query result only on the fake tuples without being discovered. As proved in [48], even a limited number of fake tuples ensures high probabilistic guarantee of completeness. The approach in [48] operates on relational databases and has been then extended to operate on XML data collections [30].

In [46] the authors propose an alternative approach based on the replication, and encryption with a different key, of a subset of the outsourced tuples. The original encrypted tuples as well as the duplicated tuples are then mixed all together and stored in the same relation. A query submitted by the client is translated into two queries determined by applying the two different encryption keys, and then they are executed on the whole encrypted relation. The client compares the results obtained by the two queries and verifies whether they both contain the same duplicate tuples. If a tuple that has been duplicated appears only in one of the two query

results, the client can immediately infer that the server has omitted at least a tuple from the result of one of these queries. Clearly, the server should not be able to determine the pairs of tuples in the outsourced relation that represent the same plaintext tuple. Like for fake tuples, the probabilistic guarantee of completeness increases with the number of duplicated tuples.

In [44], the authors present a completely different approach based on the pre-computation of *tokens* associated with a batch of queries. Before outsourcing the database, the data owner evaluates a set of queries on plaintext data and associates with each query a token computed by applying a one-way cryptographic hash function to the query result concatenated with a nonce. When the client submits a set of batch queries to the external server, it also includes one of the tokens previously computed and associated with one query in the batch. The server executes the queries in the batch and returns to the client the results of the queries along with a token and the indication of the query to which the token is associated. Such a token has to match the token previously sent from the client to the server. If this match succeeds, the client knows that the server has executed all the queries in the batch and that the result of each query is complete (because the server is not able to identify the query whose token is equal to those received from the client).

Freshness. Recent proposals address the problem of guaranteeing the *freshness* [29] of query results, meaning that queries are always executed on up-to-date data collections. In [49], the authors propose a solution for both authenticated data structures and probabilistic approaches. The basic idea behind the solution developed for the authenticated data structure approaches is to include a timestamp, which is periodically updated, in the data structure itself. If a client knows how frequently the timestamp is updated, she can check whether the received verification object (and therefore the data) is up-to-date. For instance, the signature of the root of a Merkle hash tree can be computed on the concatenation of the hash values of its children with the timestamp. The external server cannot execute queries on old versions of the data since a client, by checking the timestamp included in the signature of the root of the Merkle hash tree, can detect whether the returned query result is based on up-to-date data. The

solution proposed for probabilistic approaches adopting fake tuples [48] is based on periodically changing, in a deterministic way, the fake tuples in the dataset (i.e., new and old fake tuples are periodically added and deleted, respectively). If a client knows what are the current fake tuples in the outsourced data, it can verify whether a query result includes all and only the valid fake tuples that should be present in the instant of time when the query has been executed. At any time, the fake tuples to be inserted or deleted are generated by functions that are shared with clients.

5 Open issues

The data outsourcing scenario presents different security issues that need to be carefully addressed. Besides the problems described in this chapter, there are still other open issues and challenges that require further investigation and that we briefly describe.

- *Multiple relations.* Most of the works in the data outsourcing scenario assume that outsourced information is stored in a single relation. An interesting open issue consists in assuming that data are represented through a set of relations that can be possibly joined. The definition of confidentiality constraints needs then to be extended to capture complex requirements involving multiple relations. Analogously, the data integrity issues should be further investigated. We note that there are some attempts that consider the integrity of data resulting from the combination (join) of multiple tables. These solutions are however at a preliminary stage and may result difficult to apply in a real world scenario, where simplicity and efficiency are a must.

- *Dynamic datasets.* Solutions for protecting outsourced data typically consider static datasets since the insertion/deletion of data can possibly cause privacy breaches. For instance, the insertion of a tuple in a fragmented relation translates to the insertion of a sub-tuple in each of the physical fragments stored at the external server(s). By monitoring updates to fragments, a malicious user can reconstruct the associations among the

attributes represented in the clear in different fragments, thus violating confidentiality constraints for the new tuple. It would be then interesting to extend current approaches to support dynamic datasets.

- *Multiple owners.* In may real-world situations, the access restrictions on data arise from the collaboration of multiple parties (owners) that have a say on the data. This situation calls for novel solutions that should take into consideration the fact that the outsourced data may have multiple owners that need to collaborate to provide an adequate protection.

- *Selective write privileges.* A common assumption of all the works in the data outsourcing scenario is that write operations are permitted only to the owner of the data. This restriction however may not be applicable in many scenarios, where users need to cooperate to reach a common goal. Therefore, it would be interesting to extend current approaches for access control enforcement to support selective write privileges.

- *Instance level protection requirements.* Confidentiality constraints are defined at the schema level and describe which attributes should not be released in combination. There are however situations where associations are sensitive only when the involved attributes assume specific values. For instance, the association between the name of a specific patient and her diagnosis can be considered sensitive only if the diagnosis reveals that the patient suffers from a rare disease. Novel data protection techniques have then to be developed that take into consideration also these instance level protection requirements.

- *Private access.* Most of the solutions that guarantee confidentiality in the data outsourcing scenario are aimed at guaranteeing the privacy of remotely stored data. Another important issue that still needs to be addressed is represented by query confidentiality. In fact, a query submitted by a user can be possibly exploited for inferring sensitive information about the user. For instance, if a user accesses a medical database looking for information related to a specific disease, the server can infer that the user (or a person close to her) suffers from that disease. We note that the protection of query confidentiality

requires the protection of access patterns as well: even if each query singularly taken can be considered secure, the monitoring of a sequence of accesses may permit the server to infer sensitive information.

6 Conclusions

Data outsourcing is emerging today as a successful paradigm for the efficient management of huge data collections. As a consequence of this trend toward outsourcing, sensitive data (or data that can be exploited for linking with sensitive data) are now stored on external servers. Data confidentiality and data integrity can then be at serious risk. In this chapter, we first provided an overview of recent proposals addressing the data confidentiality issues. We described solutions based on encryption, a combination of encryption and fragmentation, and the involvement of the data owner for storing a portion of the data. We then considered the integrity issues and described different approaches that guarantee the integrity of the data in storage and the correctness, completeness, and freshness of query results. We concluded the chapter with an overview of the main open research challenges in the data outsourcing scenario.

Acknowledgments

This work was supported in part by the EU within the 7FP project "PrimeLife" under grant agreement 216483, by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4), and by the Università degli Studi di Milano within the "UNIMI per il Futuro - 5 per Mille" project "PREVIOUS".

Exercises

Exercise 1. Consider relation PATIENTS in Figure 1(a), its encrypted version in Figure 1(b), and query q = "SELECT `Diagnosis` FROM PATIENTS WHERE `DoB`<1970 AND

`County`='Orange"'. Translate q in a query that can be executed by the external server on the encrypted relation and determine the tuples returned by the server. Are there any spurious tuples?

Exercise 2. Consider the following access matrix:

	t_1	t_2	t_3	t_4	t_5	t_6
A	1	0	1	1	1	0
B	1	1	0	1	0	1
C	1	0	1	0	0	1
D	0	1	0	0	1	0
E	0	1	0	0	1	0

Determine a key derivation hierarchy by applying the heuristic algorithm described in Section 2.3.

Exercise 3. Consider relation CITIZEN(SSN, Name, DoB, ZIP, Occupation, Annual Income) and the confidentiality constraints: $c_0 = \{$SSN$\}$, $c_1 = \{$Name, DoB, ZIP$\}$, $c_2 = \{$Name, Annual Income$\}$, $c_3 = \{$DoB, ZIP, Annual Income$\}$, $c_4 = \{$Occupation, Annual Income$\}$.
Determine a correct and complete fragmentation of relation CITIZEN, according to the "non-communicating servers" fragmentation approach (Section 3.2).

Exercise 4. Consider relation CITIZEN(SSN, Name, DoB, ZIP, Occupation, Annual Income) and the confidentiality constraints: $c_0 = \{$SSN$\}$, $c_1 = \{$Name, DoB, ZIP$\}$, $c_2 = \{$Name, Annual Income$\}$, $c_3 = \{$DoB, ZIP, Annual Income$\}$, $c_4 = \{$Occupation, Annual Income$\}$.
Determine a correct and minimal fragmentation that maximizes visibility of relation CITIZEN, according to the "multiple fragments" fragmentation approach (Section 3.3).

Exercise 5. Consider relation CITIZEN(SSN, Name, DoB, ZIP, Occupation, Annual Income) and the confidentiality constraints: $c_0 = \{$SSN$\}$, $c_1 = \{$Name, DoB, ZIP$\}$, $c_2 = \{$Name, Annual Income$\}$, $c_3 = \{$DoB, ZIP, Annual Income$\}$, $c_4 = \{$Occupation, Annual Income$\}$.

Determine a correct, complete, and non redundant fragmentation of relation CITIZEN, according to the "departing from encryption" fragmentation approach (Section 3.4).

Exercise 6. Consider relation PATIENTS in Figure 1(a) and its fragmentation in Figure 8. Translate query q = SELECT `Name`, `Diagnosis` FROM PATIENTS WHERE `DoB`>1980 AND `County`≠'Napa', by applying both the parallel and sequential strategies.

Exercise 7. Consider relation PATIENTS in Figure 1(a) and its fragmentation in Figure 10. Translate query q = SELECT `Name`, `DoB` FROM PATIENTS WHERE `Diagnosis`='Diabetes' AND `County`='Napa' assuming to evaluate q on F_2^e.

Exercise 8. Consider relation PATIENTS in Figure 1(a) and its fragmentation in Figure 12. Translate query q = SELECT `Name`, `Prescription` FROM PATIENTS WHERE `Diagnosis`='Diabetes' AND `DoB`<1985 assuming to evaluate q on F_3^e.

References

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Proc. of CIDR 2005*, Asilomar, CA, USA, January 2005.

[2] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of SIGMOD 2004*, Paris, France, June 2004.

[3] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM TOCS*, 1(3):239–248, August 1983.

[4] M. Atallah, M. Blanton, N. Fazio, and K. Frikken. Dynamic and efficient key management for access hierarchies. *ACM TISSEC*, 12(3):18:1–18:43, January 2009.

[5] M.J. Atallah, K.B. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *Proc. of CCS 2005*, Alexandria, VA, USA, November 2005.

[6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proc. of Eurocrypt 2003*, Warsaw, Poland, May 2003.

[7] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, February 2005.

[8] S. Cimato, M. Gamassi, V. Piuri, R. Sassi, and F. Scotti. Privacy-aware biometrics: Design and implementation of a multimodal verification system. In *Proc. of ACSAC 2008*, Anaheim, CA, USA, December 2008.

[9] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Proc. of ESORICS 2007*, Dresden, Germany, September 2007.

[10] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Enforcing confidentiality constraints on sensitive databases with lightweight trusted clients. In *Proc. of DBSec 2009*, Montreal, Canada, July 2009.

[11] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation design for efficient query execution over sensitive distributed databases. In *Proc. of ICDCS 2009*, Montreal, Canada, June 2009.

[12] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *Proc. of ESORICS 2009*, Saint Malo, France, September 2009.

[13] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM TISSEC*, 13(3):22:1–22:33, July 2010.

[14] J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proc. of CSFW 2006*, Venice, Italy, July 2006.

[15] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proc. of CCS 2003*, Washington, DC, USA, October 2003.

[16] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proc. of VLDB 2007*, Vienna, Austria, September 2007.

[17] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM TODS*, 35(2):12:1–12:46, April 2010.

[18] A. De Santis, A.L. Ferrara, and B. Masucci. Cryptographic key assignment schemes for any access control policy. *IPL*, 92(4):199–205, November 2004.

[19] P.T. Devanbu, M. Gertz, C.U. Martel, and S.G. Stubblebine. Authentic third-party data publication. In *Proc. of DBSec 2000*, Schoorl, The Netherlands, August 2000.

[20] G. Di Battista and B. Palazzi. Authenticated relational tables and authenticated skip lists. In *Proc. of DBSec 2007*, Redondo Beach, CA, USA, July 2007.

[21] M. Gamassi, M. Lazzaroni, M. Misino, V. Piuri, D. Sana, and F. Scotti. Accuracy and performance of biometric systems. In *Proc. of IMTC 2004*, Como, Italy, 2004.

[22] M. Gamassi, V. Piuri, D. Sana, and F. Scotti. Robust fingerprint detection for access control. In *Proc. of RoboCare Workshop 2005*, Rome, Italy, May 2005.

[23] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. of STOC 2009*, Bethesda, MA, USA, May 2009.

[24] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of ICDE 2002*, San Jose, CA, USA, February 2002.

[25] H. Hacigümüs, B. Iyer, and S. Mehrotra. Ensuring integrity of encrypted databases in database as a service model. In *Proc. of DBSec 2003*, Estes Park, CO, USA, August 2003.

[26] H. Hacigümüs, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proc. of DASFAA 2004*, Jeju Island, Korea, March 2004.

[27] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of the SIGMOD 2002*, Madison, WI, USA, June 2002.

[28] K. Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939–2965, December 2009.

[29] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. of SIGMOD 2006*, Chicago, IL, USA, June 2006.

[30] R. Liu and H. Wang. Integrity verification of outsourced XML databases. In *Proc. of CSE 2009*, Vancouver, Canada, August 2009.

[31] R.C. Merkle. A certified digital signature. In *Proc. of CRYPTO 1989*, Santa Barbara, CA, USA, August 1989.

[32] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. of VLDB 2003*, Berlin, Germany, September 2003.

[33] E. Mykletun, M. Narasimha, and G. Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In *Proc. of ESORICS 2004*, Sophia Antipolis, France, September 2004.

[34] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *ACM TOS*, 2(2):107–138, May 2006.

[35] M. Narasimha and G. Tsudik. DSAC: Integrity for outsourced databases with signature aggregation and chaining. In *Proc. of CIKM 2005*, Bremen, Germany, October–November 2005.

[36] H. Pang, A. Jain, K. Ramamritham, and K.L. Tan. Verifying completeness of relational query results in data publishing. In *Proc. of SIGMOD 2005*, Baltimore, MA, USA, June 2005.

[37] H. Pang and K.L. Tan. Authenticating query results in edge computing. In *Proc. of ICDE 2004*, Boston, MA, USA, April 2004.

[38] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In R.A. DeMillo, R.J. Lipton, and A.K. Jones, editors, *Foundation of Secure Computations*. Academic Press, 1978.

[39] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, November 2001.

[40] P. Samarati and S. De Capitani di Vimercati. Data protection in outsourcing scenarios: Issues and directions. In *Proc. of ASIACCS 2010*, Beijing, China, April 2010.

[41] R.S. Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *Proc. of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow*, Dallas, TX, USA, October 1987.

[42] R.S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *IPL*, 27(2):95–98, February 1988.

[43] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2/E, 1996.

[44] R. Sion. Query execution assurance for outsourced databases. In *Proc. of VLDB 2005*, Trondheim, Norway, August–September 2005.

[45] H. Wang and Laks V. S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of VLDB 2006*, Seoul, Korea, September 2006.

[46] H. Wang, J. Yin, C. Perng, and P.S. Yu. Dual encryption for query integrity assurance. In *Proc. of CIKM 2008*, Napa Valley, CA, USA, October 2008.

[47] Z.F. Wang, J. Dai, W. Wang, and B.L. Shi. Fast query over encrypted character data in database. *CIS*, 4(4):289–300, December 2004.

[48] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *Proc. of VLDB 2007*, Vienna, Austria, September 2007.

[49] M. Xie, H. Wang, J. Yin, and X. Meng. Providing freshness guarantees for outsourced databases. In *Proc. of EDBT 2008*, Nantes, France, March 2008.

[50] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated join processing in outsourced databases. In *Proc. of SIGMOD 2009*, Providence, RI, USA, June-July 2009.

Index