Data Protection in Outsourcing using JOIN operation

RESHMA SULTANA [1], CHEEKATLA SWAPNA PRIYA [2]
[1] M.TECH STUDENT IN PYDAH ENGG ,[2] ASST PROFESSOR IN PYDAH COLLEGE OF ENGG,A.P,INDIA

ABSTRACT

Data outsourcing is an emerging paradigm that allows users and companies to give their (potentially sensitive) data to external servers that then become responsible for their storage, management, and dissemination. Although data outsourcing provides many benefits, especially for parties with limited resources for managing an ever more increasing amount of data, it introduces new privacy and security concerns. In this paper we discuss the main privacy issues to be addressed in data outsourcing, ranging from data confidentiality to data utility. We then illustrate the main re-search directions being investigated for providing effective data protection to data externally stored and for enabling their querying.

General Terms
Security, Design

Keywords
Data outsourcing, privacy, confidentiality, data protection, data fragmentation, encryption, access control

1. INTRODUCTION

In the last years, the rapid evolution of storage, processing, and communication technologies has changed the traditional ways in which data are managed, stored, and disseminated. Users are more and more interested in sharing and disseminating their personal information using the services provided by external parties (e.g., Web sites sensitive) data since the design, realization, and management of a secure system able to grant the

confidentiality of sensitive information might be very expensive. Due to the growing costs of in-house storage and management of large collections of sensitive data, since it demands for both storage capacity and skilled administrative personnel, **data outsourcing** and dissemination services have then recently seen widespread diffusion. Data out-sourcing presents important advantages: management costs are reduced and higher availability and more effective disaster protection than in-house operations are provided. On the other hand, data outsourcing opens the door to possible violations to the data and introduces therefore new issues to be addressed. Being stored externally, data are not under the control of their owners anymore, their confidentiality and integrity can therefore be at risk. That explicitly requires specific categories of sensitive information to be either **encrypted** or **kept separate** from other personally identifiable information to ensure data confidentiality. In many cases, the server it-self might not be allowed to read the actual content of the data outsourced to it for storage and management. In this case, the **honest-but-curious** server should provide effective service while operating on data that should result not intelligible to it. Honest-but-curious servers are then relied upon for ensuring availability of data and for enforcing the basic security control on the data they store. While trustworthy with respect to their services in making outsourced information available, these external servers are however trusted neither to access the content nor to fully enforce access control policy and privacy protection requirements. It is there-fore of primary

importance to provide means of protecting the confidentiality of the information remotely stored, with-out necessarily requiring trust in the subject managing the information, while guaranteeing its availability to legitimate users. A solution to these data protection issues would then allow users and companies to use a dissemination service offering strong guarantees about the protection of user privacy against both adversaries breaking into the system and the server itself. Query execution, access control enforcement, information utility and exposure to privacy breaches are all issues that require careful investigation and development of novel techniques for allowing the effective and widespread use of outsourcing services in a secure and private way.

In particular, we illustrate approaches based on: complete encryption of the data (Section 3), combination of encryption and splitting of data over two non-communicating servers (Section 4), combination of encryption and splitting of data among unlink able fragments (Section 5), and proposals based on the involvement of the owner of the data as a trusted party storing a limited amount of information (Section 6).

2. PROTECTION ISSUES IN DATA OUT-SOURCING SCENARIOS

i) users require access to the outsourced data by querying one or more external servers via a client front-end; **ii)** a **client** transforms the queries posed by users into equivalent queries operating on the data stored on the servers; **iii)** a **server** manages the outsourced data and make them available for distribution to the authorized users. A server operates on behalf of one or more **data owners** that outsource their data (or a portion of them) to it. Also, for simplicity, in the discussion and examples we assume outsourced data are stored within a relational database management system (DBMS), where data are organized in tables. We note, however, that many of the issues discussed as well as of the proposals illustrated apply to generic resources and data models.

We now describe the main issues that need to be addressed for guaranteeing proper protection and access to outsourced data.

- **Data protection.** Outsourced data are stored at external servers and outside the control of their owners. Since data might be sensitive, their content should be properly protected. Sensitive data might need to be protected from the server itself, that, while providing data storage and management should not be authorized to know the actual data content. The problem of protecting data when outsourcing them to external servers has emerged to the attention of researchers quite recently, with the introduction of the so called Database As a Service (DAS) paradigm [24, 25]. Different approaches have been proposed, typically relying on encrypting the information outsourced to the server (e.g., [7, 24, 25]) or on splitting information (fragments of the original data) across several servers or tables (e.g., [1, 9, 10]). These solutions use fragmentation, possibly combined with encryption, to break sensitive associations among outsourced data.

- **Query execution.** Since outsourced data must be protected also by the server itself, the server does not have complete visibility of the data necessary to execute possible queries independently. As a matter of fact, when data are encrypted, the DBMS running at the server is not trusted to decrypt them for the purpose of query execution; when data are split among different fragments,

the server cannot join them for responding to queries. The application of a data protection technique on the outsourced data must therefore be ac-companied by corresponding techniques enabling the execution of queries on encrypted or fragmented data. Techniques that have been investigated associate with encrypted data **indexing information**) on which queries can be executed. The challenges for indexing methods is the trade-off between precision and privacy: more precise indexes provide more efficient query execution but a greater exposure to possible privacy violations. Data protection measures must also be accompanied by proper query transformation techniques defining how queries on the original table are to be translated into queries on the encrypted or fragmented data [1, 9, 10].

- **Private access.** Data protection mentioned above re-lates to guaranteeing the privacy of the data stored at an external server. Another important issue that arises when accessing data stored at a third party is preserving the confidentiality of the query itself. The need for protecting query confidentiality can arise regardless of whether outsourced data are encrypted or not. In the first case, maintaining query confidentiality might be needed since queries themselves might be exploited by the server or by external observers for inferring information on the data content. In the second case, the reason for protecting queries arises since it is the query itself that is confidential. Consider, for ex-ample, scenarios allowing users to query external med-ical databases. The fact that a user queries the data in search for treatments for a given illness discloses the fact that the user is interested in the specific illness (and therefore the user, or a person close to her, might be suffering from it). Effective protection of query confidentiality requires not only protecting confidentiality of individual queries, but also protecting confidentiality of access patterns. In other words, it should not be possible for an observer, or for the server storing data, to infer that two queries aim at accessing the same or different data. Private access and private access pattern have recently raise attention of researchers and some directions are being investigated the definition of B-tree indexes .

- **Data integrity and correctness.** Database As a Service scenarios and proposals addressing the protection of data stored at an external server, typically assume the server to be curious (i.e., the server is not allowed to see the data content) but trustworthy, that is, relied upon for properly enforcing data storage and management. The server is then assumed reliable for properly responding to queries (provided correctness of the query translation process mentioned above). In scenarios where such a trust on the server is not applicable, there is the need to provide the data owner (or the users accessing the data) with techniques to assess the integrity and the correctness of the returned data.

  Guaranteeing integrity and correctness implies guaranteeing that the server does not improperly modify data as well as the fact that the server provides a correct response to queries (i.e., the server does not delete or modify data improperly either in storage or in query computation). Few proposals have investigated the problem of guaranteeing correctness of the data stored or returned by external outsourcing servers. Typically they are based on the use of signatures attached to tuples in the database (e.g., [31, 39]) or on chain structures (e.g., skip lists [19]) that allow the client to assess the integrity of the returned tuples.

- **Access control enforcement.** In many scenarios access to data is selective, with different users enjoying different views over the data. When data are outsourced there is therefore the problem of enforcing possible access control restrictions on it. On one hand, having the owner enforcing the access control restrictions would require the owner to mediate every query and response to filter out accesses that should not be authorized to the requesting users, causing a possible bottleneck in the system and impacting performances. On the other hand, simply outsourcing the authorization policy and its enforcement at the external server is not possible. First, the access control policy itself, like the data, might be sensitive and therefore cannot be completely disclosed. Second, access control restrictions might depend on the data content, which the server is not permitted to see. Third, completely outsourcing the management of the access control policy to the external server requires complete trust in the external server in its enforcement. There is therefore the need for developing techniques for enforcing access control in a reliable way without requesting the run time involvement of the owner. Some directions have started to be investigated relying on the combination of encryption and access control policies, towards a selective encryption enforcing itself access control. Intuitively, the key with which data are encrypted is regulated by the access authorizations holding on the data.

- **Support for selective write privileges.** Data outsourcing proposals have concentrated on the management and execution of read accesses. The assumption of limiting the large community of users to a read access while reserving the write privileges to the owner is applicable in the out-sourcing scenarios as well as in social network-like set-tings, where outsourcing is meant for data publication by owners. There are however other contexts where the consideration of read privileges only is limiting. For instance, within a multi-owner scenario selective write privileges may need to be enforced. It would then be interesting to extend current approaches for enforcing selective access to the consideration of write operations.

- **Data publication and utility.** In data outsourcing, the main goal is to give the data to an external server to avoid the burden of managing and storing them. Techniques developed for protecting data in outsourcing scenarios could also be extended and applied to data publication scenarios, where the goal is to make certain information publicly or semi-publicly available while ensuring proper protection of sensitive data. While in data outsourcing scenarios, the plaintext availability of certain data or associations among them might impact query efficiency, in data publication scenarios it im-pacts data visibility and therefore becomes of outmost importance. In data publication it is in fact crucial to guarantee a proper balance between data protection on one hand and data utility on the other hand.

- **Private collaborative computation.** Previous approaches in data outsourcing have been focused on the data exchange between an external server and a client. However, advances in communication technologies make it easy to share information among multiple servers that often need to interact to accomplish a common goal or to provide a service. This collaboration among

different servers to reach a common goal resembles the secure multi-party computation problem, where different parties need to collaborate to perform a computation on their data without however revealing the data used in the computation [21]. However, while multi-party computation approaches aim at not disclosing any of the data source content, in many scenarios selective disclosure (i.e., release of portions of data) can be applicable. In this case, the problem is to deter-mine an effective and safe execution plan for a query computation in which the servers collaborate releasing each other only information that can be disclosed for the aim of computing the query result. This problem requires the definition of approaches for the specification of the different views servers and users can have over the different data sources. Authorized views could also span data across different servers, then requiring a collaborative approach in the definition of the authorizations.

3. DATA ENCRYPTION

A first solution used for preventing a server from accessing data stored on its own machines consists in encrypting the data before outsourcing them. We now describe how data can be encrypted and accessed to minimize the workload at the client side [18]. We then discuss how to support different access privileges [15, 17].

3.1 Data model

In principle, data encryption can be performed by using either symmetric or asymmetric encryption schemas. Since however symmetric encryption is cheaper than asymmetric encryption, many proposals are based on symmetric encryption [28]. Encryption can also be applied at different granularity levels, depending on the data that need to be accessed. When data are organized as tables, encryption can be applied at the finer grain of **table**, **attribute**, **tuple**, and **element** [28]. Both table level and attribute level encryption imply the communication to the requesting client of the whole table involved in a query, as it is not possible to extract any subset of the tuples in the encrypted representation of the table. On the other hand, encrypting at the element level would require an excessive workload for data owners and clients in encrypting/decrypting data. For balancing client workload and query execution efficiency, most proposals assume that the database is encrypted at tuple level. To directly query the encrypted data (remember that confidentiality demands that data decryption must be possible only at the client side), additional **indexing information** is stored together with the encrypted database [24, 25]. Such indexes can be used by the DBMS to select the data to be returned in response to a query (see Section 3.2) and are computed starting from the plaintext values of the attributes with which they are associated. Before outsourcing a plain-text database D, each relation r over schema $R(A_1, \ldots, A_n)$ in D is therefore mapped onto an encrypted relation r^e over schema R^e (**ID**, **Etuple** , $I_1, \ldots, I_n$) in D^e , where **ID** is the pri-mary key, **Etuple** is the attribute containing the encrypted tuple, and I_i , i = 1, . . . , n, is the index associated with the i-th attribute in R (without loss of generality and for sim-plicity we assume that the encrypted relation has always an index for each attribute of the corresponding plaintext relation). For each tuple $t \in r$, there is a tuple $t^e \in r^e$ where t^e [**Etuple**] = E_k (t), with k the symmetric encryption key, and $t^e [I_i] = f(t[A_i])$, i = 1, . . . , n, with f an indexing function computing the index value $t^e [I_i]$ according to the specific indexing method adopted. Figure 1(a) illustrates an example of plaintext relation reporting information about the patients of a hospital and Figure 1(b) illustrates the corresponding encrypted relation.

The encrypted relation has Patients

SSN	Name	DoB	Zip	Treatment	Illness
123-45-6789	Alice	1969/01/01	90012	actifed	flu
652-98-3471	Bob	1965/07/23	90022	altace	heart disease
842-74-9249	Carol	1971/10/27	90010	actifed	cold
843-42-8251	Dave	1950/11/22	90005	alendronate	osteoporosis

(a)

$Patients^e$

ID	Etuple	I_S	I_N	I_D	I_Z	I_T	I_I
id1	hsh7wmdn	ρ	κ	η	δ	ν	β
id2	kjsjhc82	λ	φ	η	ρ	ϑ	α
id3	ks6b98hc	χ	μ	σ	ϑ	ϑ	α
id4	0jmkdd3	π	μ	ο	δ	ω	γ

(b)

Figure 1: An example of plaintext relation (a) and corresponding encrypted relation (b) exactly the same number of tuples as the original relation.

3.2 Query execution

The introduction of indexes makes it possible to partially evaluate any query Q at the server side, provided it is previously translated into an equivalent query operating on the encrypted data. In general, a user submits a query Q that refers to the schema of the plaintext relations in D. This query is passed to the client that maps it into a query Q_s working on the encrypted relations in D^e at the server side and a query Q_c working on the result of query Q_s at the client side. In particular, the server executes query Q_s and returns a set of encrypted tuples to the client that decrypts them and eventually discards **spurious tuples** (i.e., tuples that do not satisfy the query submitted by the user). These spurious tuples are removed by executing query Q_c. The final plaintext result is then returned to the user.

The process of transforming Q in Q_s and Q_c depends both on the indexing method adopted and on the kind of query Q. There are operations that need to be executed by the client, since the indexing method does not support them (e.g., range queries are not supported by all types of indexes) and the server cannot decrypt data. Also, there are operations that the server could execute over the index, but that require a pre-computation that only the client can per-form and therefore must be postponed in Q_c.

In the literature, di*ff*erent indexing methods have been proposed (e.g., [2, 7, 24, 27, 37]). In [24] the authors first introduce an indexing method that consists in partitioning the domain of an attribute A_i of plaintext relational schema R in a number of non-overlapping subsets of values containing contiguous values. Each partition is then associated with a unique value and the set of these values is the domain for index I_i associated with A_i . Given a plaintext tuple t in r over relational schema R, the corresponding index value is then the unique value associated with the partition to which the plaintext value $t[A_i]$ belongs. The domain of index I_i may or may not follow the same order as the one of the plain-text attribute A_i and the partitions may be chosen so that they have all the same length or contain the same number of tuples. The partition-based indexing method allows the server side evaluation of equality queries (i.e., queries with equality conditions in the where clause). Also, equality conditions involving attributes defined on the same domain can be evaluated by the server, provided that attributes are indexed using the same partition. Such methods do not easily support range queries. Since the index domain does not necessarily preserve the plaintext domain ordering, a range condition of the form $A_i \geq v$, where v is a constant value, must be mapped into a series of equality conditions operating on index I_i of the form $I_i = v_1' \vee \ldots \vee I_i = v_l'$, where $v_1' \ldots v_l'$ are the values associated with partitions that correspond to plaintext values greater than or equal to v. Note also that since the same index value is associated with more than one plaintext value, partition-based indexing usually produces spurious tuples that need to be

filtered out by the client front-end. It is easy to see that the number of spurious tuples is inversely proportional to the number of partitions since a large number of partitions increase query precision while however compromising privacy. On the other hand, a small number of partitions increases privacy but affects performance. The problem of computing an optimal partition that maximizes efficiency has been studied in [27].

Another indexing method supporting equality queries has been presented in [13]. The proposed index is based on a one-way secure hash function that takes in input the plain-text values of an attribute and returns the corresponding index values. A secure hash function satisfies important properties that turn out to be fundamental for the definition of an index. First, like the partition-based indexes, a secure hash function is deterministic, meaning that the application of a secure hash function to a given attribute value produces always the same index value, thus making easy the translation of a query Q into an equivalent query Q_s on the encrypted data. Second, a secure hash function produces collisions, meaning that different plaintext values are mapped onto the same index value. This property guarantees that even if an adversary knows the distribution of plaintext values in the original database, from the index values it is not possible to infer the corresponding plaintext values (i.e., frequency-based attacks are not applicable). Third, a secure hash function does not preserve the domain order of the attribute on which it is applied.

In addition to these two simple indexing methods, other solutions have been proposed (e.g., [2, 7, 26]). In [7] the authors present a B+-tree indexing method supporting both equality and range conditions appearing in the where clause of a query. The idea consists in using a B+-tree data structure for physically indexing data. An encrypted version of the B+-tree is then stored at the server side and is iteratively used to retrieve the desired data. The B+-tree indexing method, being order preserving, also allows the evaluation of order by and group by clauses, and of most of the aggregate operators, directly on the encrypted data. In [26] the authors present an indexing method based on **privacy homomorphism** [32]. In [2] an order preserving encryption schema (OPES) is presented to support equality and range queries as well as max, min, and count queries over encrypted data. The basic idea is that given a target distribution, the plaintext values are transformed by using an order-preserving transformation in such a way that the transformed values follow the target distribution. OPES is applicable to numeric data and is secure against cipher text-only attacks. In [36] the authors present an order preserving encryption with splitting and scaling (OPESS) schema. Splitting and scaling techniques are used to create index values so that the distribution of plaintext values is different from that of index values. Orders of plaintext values are preserved so that range queries can be easily sup-ported. Other works (e.g., [6, 20]) illustrate techniques for performing arithmetic operations (+, −, ×, ÷) on data encrypted using a privacy homomorphic encryption function. Recently a fully homomorphic encryption schema has been proposed [22] that allows the computation of an arbitrary functions over encrypted data without the decryption key.

On a different but related line of work, other proposals have been presented for searching keywords in encrypted data (e.g., [5, 35]).

Note also that when defining the indexing method for an

attribute, it is important to consider two conflicting requirements: on one hand, the indexing information should be related to the data well enough to provide for an effective query execution mechanism; on the other hand, the relation-ship between indexes and data should not open the door to inference and linking attacks that can compromise the protection granted by encryption. Different indexing methods can provide a different trade-off between query execution efficiency and data protection from inference. A deep analysis of the level of protection provided by an indexing method against inference and linking attacks is then an important aspect that has been however considered only for few proposals (e.g., [7, 27]). In [7] the authors consider the problem of quantitatively measuring the level of exposure due to the publication of indexes computed either with direct encryption or with a secure hash function. They show that even a straightforward direct encryption can provide an adequate level of protection against inference attacks, as long as a limited number of index attributes are used.

3.3 Selective access

A recent proposal for enforcing selective access to out-sourced data puts forward the idea of using selective encryption [15, 17]. The proposed approach consists in encrypting different portions of the data with different keys that are then distributed to users according to their access privileges. This idea is not new per se since it has been applied in other contexts, for example, for selectively sharing XML documents [30]. However, the problems related to the definition, management, and evolution of the authorization policy, and therefore of the corresponding encryption have been never addressed before and are instead the focus of the proposals in [15, 17]. These proposals integrate access control and encryption, meaning that the data to be out-sourced are encrypted with different keys depending on the authorizations to be enforced on the data. The authorization policy defined by the data owner is expressed through an access matrix. Such a policy is then translated into an **equivalent encryption policy** regulating which data are encrypted with which key and regulating key release to users. This translation process is performed by having in mind two important desiderata: **i)** at most one key is released to each user, and **ii)** each resource is encrypted at most once. To achieve these desiderata, the authors exploit a hierarchical organization of keys allowing the derivation of keys from other keys and public tokens [3, 4]. Basically, users with the same access privileges are grouped and each resource is encrypted with the key associated with the set of users that can access it (i.e., the set of users forming its access control list). In this way, a single key can be possibly used to encrypt more than one resource. The key derivation hierarchy used in [15, 17] exploits the hierarchy among sets of users induced by

	t_1	t_2	t_3	t_4
E	1	0	1	1
F	1	1	0	0
G	0	1	1	1
H	1	0	1	1

(a) Authorization policy

Figure 2: An example of authorization policy (a), key derivation hierarchy (b), and minimized key derivation hierarchy (c) for the relation in Figure 1(a)

the partial order relationship based on set containment ($\subseteq$). Each vertex v in the hierarchy is associated with a key k and a public label l, and each edge connecting two vertices, say v_i and v_j , is associated with a public token $t_{i,j}$ computed as $k_j \oplus h(k_i, l_j)$, with

⊕ the xor operator and h a deterministic cryptographic function [4]. Each resource is then encrypted by using the key of the vertex representing its access control list, and each user is given the key of the vertex representing herself in the hierarchy. From such a key and the public tokens, each user can derive the keys of the vertices representing groups of users containing herself. This implies that each user can decrypt all and only the resources she can access. Intuitively, the key derivation hierarchy so generated defines an encryption policy (i.e., a set of keys, a set of tokens, an association user-key, and an association resource-key) that is equivalent to the authorization policy specified by the data owner. In [17] the authors illustrate a heuristic algorithm for computing a minimal encryption policy, that is, an encryption policy where the number of tokens used (i.e., the number of edges in the key derivation hierarchy) is minimal. The rationale is to reduce the user's overhead in deriving keys maintaining only the information strictly needed to correctly enforcing an authorization policy. As an example, consider relation Patients in Figure 1(a) and

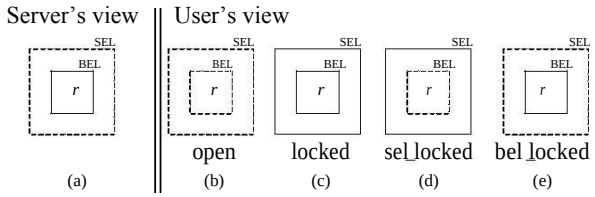

Figure 3: Possible views on resource r [15]

suppose that the tuples of this relation should be protected according to the authorization policy represented through the access matrix A in Figure 2(a). The access matrix has four columns, one for each tuple in the relation, and four rows, one for each authorized users, that is, Ellen (E), Frank (F), George (G), and Hilary (H). An entry A[u, t] in the matrix is set to 1 when user u can access tuple t; it is set to 0 otherwise. Figure 2(b) shows the key derivation hierarchy induced by the partial order relationship based on the set containment relationship defined over {E,F ,G,H }. In the figure, each vertex reports between square brackets the set of users that it represents, and the dotted edges represent the associations user-key and resource-key. It is easy to see that this key derivation hierarchy represents an encryption policy equivalent to the authorization policy in Figure 2(a). User Ellen, for example, can derive from her key (i.e., the key associated with vertex v_1) the keys associated with all vertices representing sets of users that contain E, including the keys of vertices v_{12} and v_{13} that have been used to encrypt tuples t_1, t_3, and t_4 that are all and only the tuples that Ellen can access. The key derivation hierarchy in Figure 2(b) however requires the publication of more keys and tokens than actually needed. For instance, the key associated with vertex v_{11} is not needed for enforcing the authorization policy since it is not used for encrypting any resource. Based on this observation, in [17] the authors present a heuristic algorithm that generates a key derivation hierarchy with the goal of minimizing the number of tokens to be maintained by the server to improve the efficiency of the key derivation process at the client side. The algorithm creates a minimized version of the key derivation hierarchy equivalent to the given authorization policy. Figure 2(c) illustrates a minimized key derivation hierarchy equivalent to the authorization policy in Figure 2(a).

Figure 3(a) illustrates the view of the server, which knows only the keys at the SEL and does not know the key at the BEL. Figure 3(b) illustrates the view (open) of authorized users who know both the keys at the SEL and BEL. Figures 3(c)-(e) illustrate the views of non-authorized users who do not know the keys at SEL and BEL (locked), the key at SEL (sel locked), or the key at BEL (bel locked), respectively. In [15] the authors presents an algorithm that through the combination of an encryption policy at the BEL and of an encryption policy at the SEL allows the outsourcing of the the management of the authorization policy defined by the data owner.

4. DATA FRAGMENTATION AND ENCRYPTION: NON-COMMUNICATING SERVERS

Encrypting data for storing them at the external server bears a considerable cost. More than the cost of encrypting

and decrypting data, the problem relates to the efficiency of query execution. As a matter of fact, since the server itself is not trusted for decrypting data for querying them, query execution needs to operate on indexes. As noted there is a trade-off between precision of the index (and therefore effectiveness of the queries) and privacy protection: most precise indexes allow for more efficient execution but can leak information about the indexed data, therefore opening the door to possible privacy breaches. Alternative solutions have therefore been devised trying to depart from the use of encryption to protect data since encryption might be an overdue for two main reasons. First, not all data items are sensitive and therefore should be encrypted. Non sensitive values could then be left in the clear, thus enabling the precise enforcement of selection conditions on them at the server side. Second, in many situations data themselves are not sensitive; rather their associations are sensitive. For instance, with respect to the relation in Figure 1(a), the list of patients' names and the list of illness could be made publicly available, while the association of specific illnesses with individual patients is sensitive and must be protected. Therefore, there is no need to encrypt both patients' names and illnesses if there are alternative ways of protecting their association.

4.1 Data model

The first proposal putting forward the idea of breaking associations among attributes rather than encrypting them in an outsourcing scenario is the work in [1]. In this work the authors start from the identification of the privacy requirements of the data to be outsourced. Privacy requirements are characterized as sets of attributes: singleton sets identify attributes that are sensitive per se, non singleton sets identify attributes whose association is sensitive. For instance, Figure 4 illustrates a set of privacy requirements related to relation Patients in Figure 1(a): p_0 states that Social Security Numbers individually taken are sensitive; p_1 and p_2 state that the associations between the values of attribute **Name** and of attributes **Treatment** and **Illness**, respectively, are sensitive; p_3 states that the association between the values of attributes **Treatment** and **Illness** is sensitive; p_4 and p_5 state that the association of the values of attributes **DoB** and **Zip** with the values of attributes **Treatment** and **Illness** are considered sensitive. These last two protection require

p_0 = {**SSN**}
p_1 = {**Name**,**Treatment**}
p_2 = {**Name**,**Illness**}
p_3 = {**Treatment**,**Illness**}
p_4 = {**DoB**,**Zip**,**Treatment**}
p_5 = {**DoB**,**Zip**,**Illness**}

Figure 4: Examples of protection requirements associated with the relation in Figure 1(a)

ments derive from the observation that **DoB** and **Zip** together can be exploited to infer the identity of patients (i.e., they can work as a quasi-identifier [33]), consequently their associations with other pieces of information are considered sensitive.

It is easy to see that, while simple, such a characterization of privacy requirements captures most requirements of real scenarios. To outsource data in such a way that the protection requirements identified are preserved, the approach in [1] stores the data to **two independent non-communicating servers**. Data stored at a server can be either encoded or stored in the clear. The authors propose different encoding techniques that consist in storing an attribute **A** as two separate attributes A_1 and A_2 in the two servers. For in-stance, the encrypted value of **A** can be stored at one server (i.e., $A_1 = E_k(A)$) and the encryption key at the other server (i.e., A_2=k). For simplicity, in the following we assume that encryption is used as an encoding technique. Basically, sensitive attributes (singleton constraints) need to be encrypted, while sensitive associations can be protected by splitting

(fragmenting) the involved attributes among the two servers. In addition to sensitive attributes, other at-tributes might need to be encrypted if storing them at any of the two servers in the clear would break at least one sensitive association. A relational schema R is then split into two fragments, each stored at a diff erent server. The fragments are obtained by a vertical fragmentation of the relational schema R, with some of the attributes possibly encrypted. A fragmentation of R is then a triple hF_1,F_2 ,Ei, where fragments F_1 and F_2 contain a set of attributes in the clear (including a tuple identifier to ensure lossless decomposition) and a set E of attributes encrypted (i.e., $E \subseteq F_1$ and $E \subseteq F_2$). The encrypted attributes, as well as the tuple identifier, are reported in both fragments. To guarantee protection, the set of attributes in the clear in each fragment must not be a superset of any privacy requirements, that is, for each protection requirement p_i over relational schema R, p_i $6\subseteq(F_1 - E)$ and p_i $6\subseteq(F_2 - E)$. For instance, consider relation Patients in Figure 1(a) and the protection requirements in Figure 4. Attribute **SSN** is a sensitive at-tribute (p_0) that can be protected only through encryption. Protection requirement p_1 can be satisfied by storing at-tribute **Name** in fragment F_1 and attribute **Treatment** in fragment F_2 . Protection requirement p_2 can instead be satisfied only by encrypting attribute **Illness** since it can be stored in the clear neither in F_1 nor F_2 as p_2 or p_3, respectively, would be violated. The encryption of attribute **Illness** guarantees then also the satisfaction of protection requirements p_3 and p_5 . Protection requirement p_4 can be satisfied by storing attributes **DoB** and **Zip** in fragment F_1 . The final decomposition is: hF_1 (**ID**,**Name**,**DoB**,**Zip**), F_2 (**ID**,**Treatment**),{**SSN**,**Illness**}i. Given an original relational schema R and a set P of privacy requirements, a solution decomposing the relation in two fragments as pre-scribed above always exist (in the worst case each attribute is encrypted). In general, there might exist more than one solution. A key question is therefore what is the best de-composition to use. In [1] the authors assume that the best solution is a solution that minimizes the cost of the workload being executed against the database. The characterization of the diff erent costs of decompositions is based on the use of an affinity matrix M typically used in databases, which is adapted as follows: entry M [i, j] represents the 'cost' of splitting attributes i and j (i.e., placing them in the clear in diff erent fragments); entry M [i, i] represents the cost of encrypting attribute i. The best solution is therefore a solution that minimizes the overall cost, that is, the sum of the costs of the encryption and splitting involved. The authors model the problem of finding such a solution as a hyper graph coloring problem. The hyper graph is obtained by considering a node for each attribute, where node i is associated with weight M [i, i] and each edge hi, ji is associated with weight M [i, j]. Also, each privacy requirement is represented as an hyper-edge connecting the involved attributes. Encrypting an attribute corresponds to deleting the corresponding node. Storing an attribute in the clear on a server corresponds to coloring the node with the color of the server. The problem is therefore to determine a 2-coloring of the graph in such a way that the sum of the weights of deleted nodes and of bichromatic edges is minimized. Clearly the problem is NP-hard. The authors then propose diff erent heuristics for its solution, which make use of approximate min-cuts and of approximate weighted set cover as basic techniques.

4.2 Query execution

Since the original relation is split among the two external servers, query execution may need to access information at both servers and properly combining it. Reformulating a query over the two fragments is rather straightforward, as it reduces to substituting the join among the two fragments ($F_1 \lhd\rhd F_2$) in place of the original relational schema R in the query plan. The query plan can then undergo traditional query optimization, with minor modifications to account for attribute fragmentation. For instance, projections may be pushed down to both fragments, taking care not to project out tuple identifiers necessary for the join; selection conditions involving an individual attribute may be pushed down to the corresponding fragment (if the at-tribute appears in the clear); selection conditions involving more attributes may be pushed down to the fragment containing them in the clear (if any). Once the query plan is optimized, the physical plan determines how the query execution is partitioned across the two servers and the client. The basic partition of the plan is straightforward: all operations above the top-most join have to be executed at the client side; all operations under the join and above F_i are executed by the server storing F_i . In some cases, it may be possible to push all operators to either F_1 or F_2 , thus eliminating

the need for a join. Otherwise, the join must be executed. There are essentially three options for executing the sub-queries and the join. The first option is to execute the sub-queries on F_1 and F_2 in parallel and join the results at the client side. The second option and third option are to execute a sub-query at one of the server first, and to per-form a semi join of the returned tuple identifiers with the

```
select Name,Illness
from   Patients
where DoB<1970/01/01 and Treatment like 'actifed'
```

(a) Original query Q

Q_1

```
select ID ,Name,Illness
from   F1
where DoB<1970/01/01
```

Q_2

```
select ID ,Illness as k
from   F2
where Treatment like 'actifed'
```

(b) Sub-queries at the servers

```
select Name, Decrypt(Res_Q1
.Illness, k) as Illness from Res_Q1 ,
Res_Q2
where Res_Q1 .ID = Res_Q2 .ID
```

(c) Query at the client

Figure 5: An example of query translation in the non-communicating servers scenario

fragment on the other server in addition to executing the sub-query on it. The first option is more expensive, since it requires more data to be transmitted from the servers to the client and the execution of the join at the client side. The second and third options potentially enjoy a lower cost (de-pending on the selectivity of the sub-query executed first) but imply a sequential computation and a possible risk of privacy breaches. In fact, it implies disclosing to one of the servers the tuple identifiers that satisfy the condition on the other server. Even assuming the query is not known (as otherwise privacy would be compromised), the fact that some tuple identifiers enjoy some common characteristics may be exploited for withdrawing inferences on possible values in the tuples.

As an example of query execution, suppose that a user submits query Q in Figure 5(a) that returns the name and illness of all patients born before 1970 and whose treatment is Actifed. Suppose also that the encrypted values of at-tribute **Illness** are stored at the first server and the encryption key at the second server. Query Q is translated into two sub-queries Q_1 and Q_2 (see Figure 5(b)) that are executed over fragments F_1 and F_2 , respectively. Query Q_1 retrieves from the first server the tuple identifier (**ID**), the encrypted attribute **Illness**, and the **Name** of patients born before 1970. Query Q_2 applies the selection on attribute **Treatment** and returns the tuple identifier (attribute **ID**) and attribute **Illness** (renamed as k) corresponding to the key used for encrypting the values of attribute **Illness** in Patients. Finally, the client executes a query that performs a join between the results of queries Q_1 and Q_2, denoted Res_{Q1} and Res_{Q2} , respectively, and decrypts attribute Res_{Q1} .**Illness** using the key retrieved from Q_2 (see Figure 5(c)).

5. DATA FRAGMENTATION AND ENCRYPTION: UNLINKABLE FRAGMENTS

While presenting an interesting direction, the approach in [1] suffers from two major limitations. First, privacy relies on the absence of communication between the two servers,which have to be completely unaware of each other. This assumption is clearly too strong and difficult to enforce in real environments. A collusion among the servers (or the users accessing them) easily breaches privacy. Second, the assumption of two servers limits the number of associations that can be solved by fragmenting data, often forcing the use of encryption. In [10] the authors address these limitations while exploiting the combined use of fragmentation and encryption proposed in [1]. We now describe this proposal more in details.

5.1 Data model

The starting point of the problem, that is, a relational schema R and a set of privacy requirements (called confidentiality constraints in [10]) are the same as in [1]. Differently from [1], in [10] the authors assume that multiple fragments can be created and stored at diff erent servers or even at the same server. Rather than relying on the storing servers not knowing each other, the approach in [10] relies on the fact that fragments are guaranteed to be not linkable (i.e., it is not possible for parties diff erent from the client to reconstruct the original relation and determine the sensitive values and associations). Encryption is applied at the at-tribute level, that is, it involves an attribute in its entirety. Encrypting an attribute means encrypting (tuple by tuple) all its values. To protect encrypted values from frequency attacks [34], a salt is applied to each encryption. Fragmentation, like encryption, applies at the attribute level, that is, it involves an attribute in its entirety. Fragmenting means splitting sets of attributes so that they are not visible together, that is, the associations among their values are not available without access to the encryption key. While singleton constraints can be solved only by encryption, every association constraint could be solved by either: **i)** encrypt-ing any (one success) of the attributes involved in the constraint, so to prevent joint visibility, or **ii)** fragmenting the attributes involved in the constraint so that they are not visible together. For instance, with respect to relation Patients in Figure 1(a) and the protection requirements in Figure 4, a possible fragmentation, denoted F, is {{**Name**,**DoB**,**Zip**}, {**Illness**}, {**Treatment**}}. At the physical level the original relation is represented as a set of physical fragments each containing: a salt (also exploited as tuple identifier), a set of attributes of R in the clear, and an encrypted attribute corresponding to the encrypted subtuple of all the attributes that are not represented in the clear. Privacy is guaranteed by requesting that: **i)** no fragment contains in the clear all the attributes appearing together in a confidentiality constraint and **ii)** fragments do not have attributes in common (i.e., they cannot be linked). Note that the use of a salt guarantees that the encrypted values cannot be used for linking. Figure 6 illustrates the physical fragments corresponding to fragmentation F ={{**Name**,**DoB**,**Zip**}, {**Illness**}, {**Treatment**}} of relation Patients in Figure 1(a).

Since the availability of attributes in the clear in a fragment permits an efficient execution of queries, fragmentation is considered to be preferred over encryption whenever possible. In other words, association constraints are solved via fragmentation, and encryption is limited to those attributes that are sensitive by themselves (i.e., singleton constraints). Similarly to what noted in [1], diff erent fragmentations can exist, all limiting encryption to sensitive attributes but differing in how the attributes are distributed in the fragments

F_1

Salt	Enc	Name	DoB	Zip
s_1	α	Alice	1980/01/01	90012
s_2	β	Bob	1965/07/23	90022
s_3	γ	Carol	1971/10/27	90010
s_4	δ	Dave	1950/11/22	90005

F_2

Salt	Enc	Illness
s_5	ϱ	flu
s_6	ε	heart disease
s_7	ζ	cold
s_8	η	osteoporosis

F_3

Salt	Enc	Treatment
s_9	θ	actifed
s_{10}	ϑ	altace
s_{11}	ι	actifed
s_{12}	κ	alendronate

Figure 6: Physical fragments for the relation in Fig-ure 1(a) and enforcing the requirements in Figure 4

and/or in the number of fragments. Again, the goal is to determine a solution that provides minimality. In [10] the authors assume minimality to be characterized by the num-ber of fragments in a fragmentation and investigate therefore the problem of determining a fragmentation with a mini-mum number of fragments. Since the problem is NP-hard, the authors introduce an alternative definition of minimality and assume that a solution is minimal if merging any two fragments would break at least a confidentiality constraint. They then propose a heuristic approach to its solution. The minimization of the number of fragments exploits the basic principle according to which the presence of a high number of attributes in the clear permits an efficient execution of queries. While this principle may be considered acceptable in many situations, diff erent combinations of attributes can be accessed all together a

different number of times, depend-ing on the queries executed on the data. In [10] the authors then acknowledge the need for keeping together some specific attributes according to the queries that are frequently executed on the data. Given a query Q and a fragmentation F, the execution cost of Q varies according to the specific fragment used for computing the query. This implies that, with respect to a specific query workload, different fragmentations may be more convenient than others in terms of query performance.

To take into consideration the query workload in the fragmentation process, the authors exploit the concept of at-tribute affinity, where attribute affinity is also a measure of how strong the need of keeping the attributes in the same fragment is (i.e., what is the cost of splitting the attributes in different fragments). Attribute affinity is then naturally extended to fragments and fragmentations. Intuitively, the affinity of a fragment is the sum of the affinity of the different pairs of attributes in the clear in the fragment; the affinity of a fragmentation is the sum of the affinity of its fragments. Fragmentations that maintain together attributes with high affinity are to be preferred. Again, the problem is NP-hard, and the authors present a heuristic approach to its solution.

In [8] the authors go a step further in aiming at characterizing what fragmentation can provide best with respect to a given workload and, instead of characterizing the workload with the affinity matrix, they assume that a query workload is given as a set of queries together with their frequency of execution. The authors then present a query cost model that is used to evaluate the cost of a query, and therefore of a query workload, against a fragmentation and introduce the problem of determining a fragmentation that minimizes

```
select Name,Illness from
Patients
where DoB<1970/01/01 and Treatment like 'actifed'
```

(a) Original query Q

```
select Salt,Enc,Name
from   F1
where DoB<1970/01/01
```

(b) Query operating on fragment F_1

```
select Name,Illness
from Decrypt(Res_QF1 , k) where
Treatment like 'actifed'
```

(c) Query operating at the client

Figure 7: An example of query translation in the unlinkable fragments scenario

the cost of executing the given query workload. Their formulation of the problem is based on the definition of the space of the different fragmentation and on its organization as a lattice (with bottom and top elements the extreme fragmentations represented by putting all attributes in the same fragments or each attribute in a different fragment, respectively). Noting the monotonicity of the fragmentation cost over the lattice, the authors propose a heuristic algorithm that partially visits the lattice, following a top-down strategy to compute a locally minimal fragmentation that, as proved by experimental results has a cost near to the optimum.

5.2 Query execution

Since each fragment contains all the original attributes, in either the clear or encrypted form, it is sufficient to access a fragment (any fragment) for executing a query, although different fragments may differ with respect to the efficiency (i.e., the cost) of running the query. In [10] the authors ad-dress the translation and execution of select-from-where queries. Query execution is rather simple. The only observation is that a selection conditions can be pushed down to a fragment only if all involved attributes appear in the fragment in the clear, otherwise it needs to be executed at the client (returning all the attributes needed for evaluation). Hence, the execution of a query on a given fragment re-quires translating the query into two different queries. First, a query, executed at the external server on the stored fragment that evaluates all selection conditions that operate on attributes that are in the clear in the fragment and returns the requested attributes as well as the salt (attribute **Salt**) and the encrypted field (attribute

Enc), if the query needs to evaluate attributes that are encrypted in the fragment. Then, a query is executed at the client: the returned data are decrypted, the remaining conditions are evaluated, and the attributes requested as result are returned. As noted, any query could be executed on any fragment although different fragments may diff erent with respect to the query cost. In particular, it is better to execute the query on a fragment that allows the most selective conditions to be pushed down to the fragment. As an example, consider again the query returning the name and illness of patients born before 1970 and whose treatment is actifed (see Figure 7(a)). Sincefragment F_1 contains attribute **DoB** in the clear, which we assume to be more selective than attribute **Treatment**, F_1 is chosen for query evaluation. The server storing fragment F_1 then executes a query that selects the tuples that satisfy the condition on **DoB** and returns attributes **Salt**, **Enc**, and **Name** (see Figure 7(b)). Finally, the client decrypts the con-tent of attribute **Enc** of the tuples returned by the server and on the decrypted tuples executes a query that retrieves those satisfying the condition on attribute **Treatment**. Figure 7(c) illustrates the query executed at the client side, where k is the decrypting key and Res_{QF1} denotes the result of the query in Figure 7(b).

6. DATA FRAGMENTATION WITH OWNER INVOLVEMENT

Proceeding along the directions of minimizing the use of encryption, in [9] the authors put forward the idea of completely departing from encryption and adopt fragmentation as the only means of protecting privacy when outsourcing data. The rational for the assumption that data should not be encrypted is that encryption is sometimes considered a too rigid tool, delicate in its configuration, and requiring careful management to fulfill its potential. Systems protecting sensitive information based on an extensive use of encryption suffer from significant consequences due to both the compromise and loss of keys. In the real world, key management, particularly the operations at the human side, is a difficult and delicate process. Also, as already noted, while the computational cost of symmetric encryption for modern computer architectures is usually negligible, the presence of encryption often causes an increase in the computational load, affecting the performance of query execution.

6.1 Data model

In [9] the authors depart from encryption by involving the data owner in storing, and managing, a small portion of the data, while delegating the management of all other data to the external server. The management of a small portion of data is considered an advantage with respect to the otherwise required encryption. The need for the data owner to maintain control on part of the data is to avoid exposing sensitive attributes or associations externally. Sensitive attributes are maintained at the owner side. Sensitive associations are protected by ensuring that not all the attributes in an association are stored externally. In other words, for each sensitive association, the owner should locally store at least one attribute. The original relational schema R is split in two fragments: F_o , stored at the data owner, and F_s , stored at the external server. To correctly reconstruct the content of a relation r over schema R, at the physical level, F_o and F_s have a common tuple identifier that corresponds to the primary key of R, if it is not sensitive, or can be an attribute that does not belong to the schema of R and that is added to F_o and F_s after the fragmentation process. A fragmentation $hF_o, F_s i$ is considered correct if it satisfies the following conditions: **1)** all attributes in R should appear in at least one fragment, to avoid loss of information; **2)** the external fragment should not violate any confidentiality constraint. Note that this condition applies only to F_s since F_o is under the data owner control and therefore is accessible only to authorized users. Also, a fragmentation should be non redundant, that is, the two fragments should have

F_o

ID	SSN	Treatment	Illness
id_1	123-45-6789	actified	flu
id_2	652-98-3471	altace	heart disease
id_3	842-74-9249	actified	cold
id_4	843-42-8251	alendronate	osteoporosis

F_s

ID	Name	DoB	Zip
id_1	Alice	1980/01/01	90012
id_2	Bob	1965/07/23	90022
id_3	Carol	1971/10/27	90010
id_4	Dave	1950/11/22	90005

Figure 8: An example of physical fragments with owner

involvement

no attribute in common. While not needed for preserving privacy, non redundancy avoids unnecessary storage at the data owner side (there is no need to maintain information that is outsourced); it also avoids usual replica management problems. Figure 8 illustrates a possible fragmentation of relation Patients in Figure 1(a) that satisfies the protection requirements in Figure 4.

Similarly to previous approaches, given a relation and a set of protection requirements (confidentiality constraints) on it, the problem is to determine a fragmentation that pro-vides best, where 'best' is to be defined with respect to a cost for the owner of executing queries against the fragmented data. The starting observation is that storage and computational resources offered by the external server are considered, for a given level of availability and accessibility, less expensive than the resources within the trust boundary of the owner. The owner has then a natural incentive to rely as much as possible, for storage and computation, on the external server. In the absence of confidentiality constraints, all data would then be remotely stored and all queries would be computed by the external server. In the case of confidentiality constraints, the owner internally stores some attributes, and consequently is involved in some computation.

In [9] the authors discuss several metrics (and corresponding weight functions to be minimized) that could be used to characterize the quality of a fragmentation, and therefore to determine which attributes are stored at the owner side and which attributes are outsourced at the external server. The diff erent metrics may be applicable to different scenarios, depending on the owner's preferences and/or on the specific knowledge (on the data or on the query workload) available at design time. The authors consider four possible scenarios, in increasing level of required knowledge. The first two scenarios support measuring storage, while the latter two scenarios support measuring computation.

- **Min-Attr** . Only the relation schema (set of attributes) and the confidentiality constraints are known. The only applicable metric aims at minimizing the storage

  required at the owner side by **minimizing the number of attributes** in F_o .

- **Min-Size**. Besides the mandatory knowledge of the relation schema and confidentiality constraints on it, the size of each attribute is known. In this case, it is possible to produce a more precise estimate of the storage required at the owner side, aiming at **minimizing the**

  physical size of F_o , that is, the actual storage required by its attributes.

- **Min-Query** . In addition to the relation schema and the confidentiality constraints, a representative profile of the expected query workload is known. The profile defines, for each query, the frequency of execution and the set of attributes evaluated by its conditions. Here, the goal is to minimize the **number of query executions that require processing at the owner side**, producing immediate benefits in terms of the reduced level of use of the more expensive and less powerful computational services available at the owner.

- **Min-Cond** . In addition to the relation schema and the confidentiality constraints, a complete profile of the expected query workload is known. The complete pro-file assumes that the specific conditions (not only the attributes on which they are evaluated) appearing in each query are known. The precise characterization

of the workload allows the definition of a metric to minimize the **number of conditions that require processing at the owner side**. Note that the minimization of the conditions executed at the owner side has a direct relationship with the minimization of the traffic needed for receiving results of the portion of queries outsourced to the external server. As a matter of fact, minimizing the conditions executed by the owner is equivalent to maximizing the conditions outsourced to the external server, and therefore delegating to it as much computation as possible. In fact, since the result of evaluating a condition on a relation is a smaller relation, the greater the number of conditions outsourced to the external servers, the smaller will be the corresponding results to be received in response.

In [9] the authors provide a uniform modeling of the fragmentation problem, encompassing the diff erent metrics above, which can be simply represented by the definition of a proper weight function input to the minimization problem. The minimization of the cost of involving the owner (either for storage or computation) is NP-hard (it reduces to the minimum hitting set in its simplest form of minimizing the number of attributes). The authors then provide a heuristic algorithm for the computation of a solution that guarantees minimality (i.e., moving any attribute from F_o to F_s would violate at least one constraint). Also, according to experiments, the returned solution well approximates the optimum.

6.2 Query execution

Like for the fragmentation approach based on two non-communicating servers, query execution may need to access the information stored on the two fragments F_o and F_s . A select-from-where query Q defined over the original relational schema R is then translated into queries operating on the two fragments. This translation process can follow two basic strategies: **client-first** and **server-first** .

With the client-first strategy a query Q_o is first executed at the client side. Query Q_o is obtained from the original query Q as follows. The select clause of query Q_o contains attribute **ID** since it is needed to perform a join operation between the result of Q_o and F_s ; other attributes therefore cannot appear in the select clause because they cannot be


```
select Name,Illness from
Patients
where DoB<1970/01/01 and Treatment like 'actifed'
```

(a) Original query Q

Client-first

```
select ID
from    F_o
where Treatment like 'actifed'
```

(b) Query Q_o

```
select ID ,Name
from    Res_Qo ,F_s
where Res_Qo .ID =F_s .ID and
        DoB<1970/01/01
```

(d) Query Q_s

```
select Name,Illness
from    F_o ,Res_Qs
where F_o .ID =Res_Qs .ID
```

(f) Query Q_{os}

Server-first

```
select ID ,Name
from    F_s
where DoB<1970/01/01
```

(c) Query Q_s

```
select Name,Illness
from    F_o ,Res_Qs
where F_o .ID =Res_Qs .ID and
                Treatment like 'actifed'
```

(e) Query Q_o

Figure 9: An example of query translation in the owner involvement scenario

communicated to the server. The where clause of Q_o contains all conditions of Q that involve attributes stored in F_o only since their evaluation can be performed only by the data owner. The client executes Q_o and sends to the server a query Q_s operating on the join between F_s and the result of Q_o . The select clause of Q_s contains all attributes of F_s appearing in the select clause of the original query Q and all attributes in F_s appearing in conditions that are involved in a comparison with attributes in F_o . The conditions in the where clause of Q_s are those appearing in the original query Q and involving attributes stored in F_s only and that therefore can be evaluated by the server. The re-sult of query Q_s is then sent back to the client, which further refines the result possibly executing another query Q_{os} on the join between F_o and the result of query Q_s . Query Q_{os} applies the conditions in Q that involve at the same time at-tributes stored in F_o and F_s . Note that if the server knows the original query Q, the client-first strategy cannot be used since the server infer the tuples that satisfy the conditions in the where clause of Q and that involve attributes stored in F_o only.

With the server-first strategy a query Q_s is first executed at the server side. The result of Q_s is then further refined at the client side. Query Q_s is obtained from the original query Q as follows. The select clause of Q_s contains attribute **ID** needed for performing the join between the result of Q_s and F_o , and all attributes in F_s appearing in the select clause of Q or that appears together with attributes in F_o in conditions in the where clause of Q (these conditions can therefore be evaluated by the data owner only). The where clause of Q_s contains all conditions appearing in the where clause of Q that involve attributes stored in F_s only. The server executes Q_s and returns the corresponding result to the client that performs the join with F_o and removes the tuples that do not satisfy the conditions in the where clause of Q and that involve attributes in F_o only or attributes in both F_o and F_s .

Figure 9 illustrates an example of query execution operating on the fragments illustrated in Figure 8 according to both the client-first (left-hand side) and server-first (right-hand side) strategies. Note that in the client-first strategy, the last query Q_{os} (Figure 9(f)) has only a join condition in the where clause since the original query Q does not have any condition that involves both attributes in F_o and attributes in F_s .

7. CONCLUSIONS

Effective adoption of data outsourcing solutions as well as effective information sharing and dissemination can take place only if data owners can be assured that, while releasing or storing information externally, disclosure of sensitive information is not a risk. Data protection and privacy in emerging storing and sharing scenarios is far from been a trivial problem and requires the investigation of new issues and the design of technological solutions to address them. This paper has discussed problems to be addressed and il-lustrated some emerging directions introducing novel data protection approaches in outsourcing scenarios.

8. 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 the Second Biennial Conference on Innovative Data Systems Research (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 ACM SIGMOD 2004**, Paris, France, June 2004.

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

[4] M. Atallah, K. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In **Proc. of the 12th ACM Conference on Computer and Communications Security (CCS 2005)**, Alexandria, USA, November 2005.

[5] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In **Proc. of the 4th Theory of Cryptography Conference (TCC 2007)**, Amsterdam, The Netherlands, February 2007.

[6] C. Boyens and O. Gunter¨. Using online services in untrusted environments - a privacy-preserving architecture. In **Proc. of the 11th European Conference on Information Systems (ECIS 2003)**, Naples, Italy, June 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 Transactions on Information and System Security (TISSEC)**, 8(1):119–152, February 2005.

[8] 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 the 29th International Conference on Distributed Computing Systems (ICDCS 2009)**, Montreal, Quebec, Canada, June 2009.

[9] 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 the 14th European Symposium On Research In Computer Security (ESORICS 2009)**, Saint Malo, France, September 2009.

[10] 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 Transactions on Information and System Security (TISSEC)**, 2010. (to appear).

[11] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-Anonymity. In T. Yu and S. Jajodia, editors, **Secure Data Management in Decentralized Systems**. Springer-Verlag, 2007.

[12] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. In **Proc. of the 34th International Conference on Very Large Data Bases (VLDB 2008)**, Auckland, New Zealand, August 2008.

[13] 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 the 10th ACM Conference on Computer and Communications Security (CCS 2003)**, Washington, DC, USA, October 2003.

[14] T. K. Dang. Oblivious search and updates for outsourced tree-structured data on untrusted servers. **International Journal of Computer Science & Applications**, 2(2):67–84, 2005.

[15] 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 the 33rd International Conference on Very Large Data Bases (VLDB 2007)**, Vienna, Austria, September 2007.

Academic Press, Orlando, FL, USA, 1978.

Authors:

Reshma sultana m.tech student in pydah engg and tech. interested areas are data mining, data bases, networks and web technologies

cheekatla swapna priya Asst professor in pydah college of Engg. and Tech 7 1/2 experience in both postgraduate and under graduate courses. Interested in data mining and networks.