Selective and Fine-Grained Access to Data in the Cloud

Sabrina De Capitani di Vimercati, Sara Foresti, and Pierangela Samarati

Abstract This chapter surveys some of the research results related to the protection and efficient access to data stored and managed by external cloud servers. We first provide an overview of the security and privacy problems and challenges that need to be considered, and then illustrate emerging approaches for protecting data externally stored, and for enforcing fine-grained (queries) and selective (access control) accesses on them. Finally, we show how the combined application of the solutions discussed may introduce privacy problems that should be carefully considered.

1 Introduction

Emerging paradigms like data outsourcing and cloud computing have attracted the attention of the research and industrial communities thanks to their advantages in terms of reduced costs for IT resources, increased storage, flexibility in resource management, and higher scalability. These advantages however do not come for free. In fact, these emerging paradigms also introduce a number of privacy and security risks that may represent a serious obstacle for their wide development and for their acceptance by users and companies. Security and privacy may relate to different aspects, including resources, data and network isolation, attacks to the cloud servers, compliance with laws and regulations, reliability of applications and services, protection of the confidentiality and integrity of data, and data availability (e.g., [11, 38, 39, 44]). In this chapter, we will provide an overview of the problems and solutions related to the proper protection of the confidentiality of the data and to the efficient access to them. These problems become quite complex in a cloud scenario since users release and store their data on external servers that are outside

Sabrina De Capitani di Vimercati · Sara Foresti · Pierangela Samarati
Università degli Studi di Milano – Dipartimento di Informatica
Via Bramante 65, 26013 Crema, Italy
e-mail: *firstname.lastname*@unimi.it

their control. Also, the advances in the Information and Communication Technologies (ICTs), including the possibility of combining and analyzing more information from several data sources, intensify the data protection problem.

The protection of potentially sensitive data stored and managed by external cloud servers poses interesting challenges. In fact, cloud servers can be characterized by different levels of trust, ranging from *honest-but-curious* servers, meaning that they are trusted for the management of the data but cannot know (access) the data they store, to servers that may intentionally behave improperly in the storing and processing of the data. Data are therefore encrypted by the data owner before their storage in the cloud. Since cloud servers cannot decrypt data, there is the problem of defining techniques (e.g., indexes) for enforcing fine-grained retrieval of the data without compromising their privacy. However, techniques that support effective and efficient accesses to the outsourced data are not enough. In fact, if the server (or a generic observer) monitors the accesses by users, it may be able to draw inferences on which data have been accessed. Also, the presence of multiple users who rely on external storage for making their data available to others, introduces the problem of enforcing selective (read and write) access to the outsourced data.

In this chapter, after a brief overview of the different security and privacy problems that can arise in a cloud computing scenario, we survey and discuss research results related to the protection of the privacy of outsourced data, and on the fine-grained and selective retrieval of data. We also show that the combination of techniques addressing a specific problem can cause privacy breaches. The remainder of the chapter is organized as follows. Section 2 provides an overview of the main security and privacy risks in a cloud scenario. Section 3 illustrates some approaches and open issues related to the protection of data confidentiality, indexing for query support, and selective access. Section 4 describes how the combination of indexes for query support and fragments for data confidentiality can cause leakage of confidential information. Section 5 describes how the combination of indexes and selective encryption may allow unauthorized users to infer (or reduce their uncertainty on) information that they are not authorized to access. Finally, Section 6 provides our conclusions.

2 Security and privacy in the cloud

The security and privacy problems that arise when data are stored at external servers have been the subject of many studies (e.g., [22, 31, 37]). Depending on the considered aspect, the security and privacy problems can be related to: *i)* the privacy of users; *ii)* the privacy and integrity of data storage; *iii)* the privacy and integrity of queries; and *iv)* the secure and private data computations involving multiple providers. Figure 1 illustrates the reference cloud scenario where users interact with external cloud servers for accessing data and services, and different cloud servers collaborate for offering a service or responding to a query. In the remainder of this

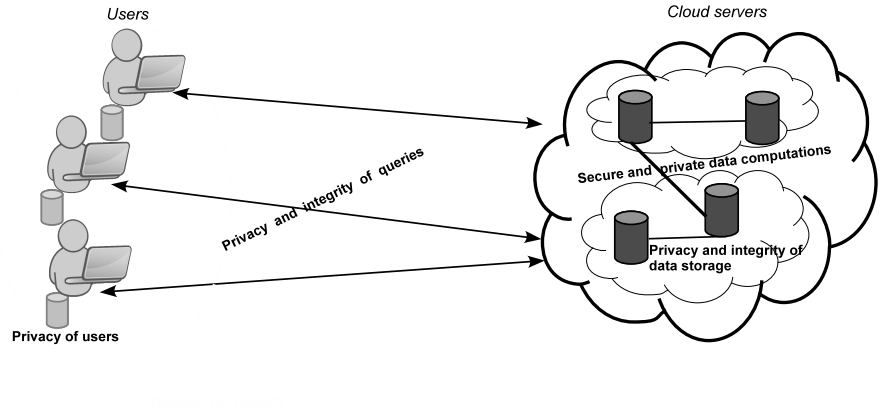


Fig. 1 Reference cloud scenario

section, we provide a description of each of the four categories of security and privacy problems mentioned above.

Privacy of users. Cloud services allow users to access applications and data on demand every-time they need. To successfully complete the required access, users may be asked to provide some information while however wishing to protect their identities for privacy reasons. For instance, a user can be interested in querying a cloud server for collecting information about a given illness without revealing her identity to avoid possible correlations between the illness and herself or a person close to her. The techniques developed for supporting anonymous communication between parties and attribute-based access control can be helpful in protecting the privacy of the users. In fact, anonymous communication techniques allow users to communicate on the Internet without revealing their identities [9], meaning that an observer cannot trace who is communicating with whom, or who is interacting with which server or searching for which data. Attribute-based access control solutions allow users to access resources or data without reveling their identities [13]. The idea is that, instead of declaring their identities, users prove that they satisfy the conditions needed for the access. To this purpose, a user can disclose a *credential* (a set thereof) certifying the information necessary for the access. The server verifies whether the credential is valid and whether the information it certifies satisfies the policy regulating access to the resource. The research community has also devoted considerable attention to the use of *anonymous credentials* [16] for access control (e.g., [4]). An anonymous credential allows a user to make statements about attribute values, maintaining the values private. For instance, anonymous credentials permit to selectively release a subset of the properties in a credential or to prove that they satisfy some conditions, without revealing any information about their values. Anonymous credentials can be at the basis of a new generation of access control policy languages that can be particularly suited to open and dynamic scenarios like the cloud.

Recently, some proposals have started to address the problem of regulating the release of users' personal information according to privacy preferences expressed by the users themselves. These proposals have introduced models relying on user preferences that permit to associate a higher or lower sensitivity with the combined release of a set of properties/credentials (e.g., [5, 6, 7, 40, 53]). For instance, a user may consider the joint release of her name and credit card number more sensitive than the release of each information singularly taken. Although these solutions represent a first step towards the definition of a comprehensive approach for the protection of users' privacy, there are still several open issues: the development of user-friendly approaches for expressing privacy preferences; the ability of defining privacy preferences that depend on the context; and the integration of these approaches with server-side solutions supporting fine-grained policy disclosure, which permit the server to obfuscate the portions of its policies considered sensitive, while providing the user with enough information for releasing the information necessary to possibly gain access (e.g., [8]).

Privacy and integrity of data storage. When data are outsourced to an external server that is outside the control of the data owner, the protection of the confidentiality and of the integrity of the data, as well as the efficient access to them become clearly of paramount importance. In this context, the research community has been very active and produced advancements in several areas: solutions for *protecting data confidentiality* (e.g., encryption and fragmentation [1, 21, 37]); *indexes* for supporting queries (e.g., [17, 37]), solutions for supporting *selective access* to outsourced data (e.g., [23]), solutions for ensuring *data integrity* (e.g., signatures [14, 35, 43]). These approaches typically consider a scenario where a *data owner* outsources her data to an *external server* that can be trusted to properly manage the data, making them available to requesting *users*, but it is not trusted to read the content of the data it stores (i.e., *honest-but-curious* server). The outsourced data can be of any type, including files and relational tables. In the remainder of this chapter, for simplicity and without loss of generality, we will assume that the outsourced data are organized in a single relation r, stored in a (distributed) relational database. Relation r is defined over relational schema $R(a_1,\ldots,a_n)$, with attribute a_i defined over domain D_i, $i = 1,\ldots,n$. The presentation of solutions and issues related to the protection of the privacy of outsourced data will be the subject of the following sections.

Privacy and integrity of queries. Accessing information from external cloud servers and performing queries over outsourced data introduce several privacy and integrity issues. Existing data management architectures typically assume that the data obtained from distributed parties have not been tampered with, and are available only to authorized parties. Such assumptions do not apply anymore in cloud scenarios, where multi-tenant infrastructures orchestrate different services. Assurances on the fact that the privacy of the queries is preserved and that computations on data are processed in the expected way (integrity and verifiability) are becoming more and more important. In fact, there is an increasing need for novel techniques that support not only data privacy, but also the privacy of the accesses that users

make on such data. This problem has been traditionally addressed by Private Information Retrieval (PIR) proposals (e.g., [18]), which provide protocols for querying a database that prevent the external server from inferring which data are being accessed. PIR solutions however have high computational complexity, and alternative approaches have been proposed. These novel approaches rely on the Oblivious RAM structure (e.g., [33, 47, 48]) or on the definition of specific tree-based data structures combined with a dynamic allocation of the data (e.g., [29, 30]). The goal is to support the access to a collection of encrypted data while preserving access and pattern confidentiality, meaning that an observer can infer neither what data are accessed nor whether two accesses aim to the same data. Besides protecting access and pattern confidentiality, it is also necessary to design mechanisms for protecting the integrity and authenticity of the computations, that is, to guarantee the correctness, completeness, and freshness of query results. Most of the techniques that can be adopted for verifying the integrity of query results operate on a single relation and are based on the idea of complementing the data with additional data structures (e.g., Merkle trees) or of introducing in the data collection fake tuples that can be efficiently checked to detect incorrect or incomplete results (e.g., [41, 46, 50, 51, 52]). Interesting aspects that need further analysis are related to the design of efficient techniques able to verify the completeness and correctness of the results of complex queries (e.g., join operations among multiple relations, possibly stored and managed by different cloud servers with different levels of trust).

Secure and private data computations. More and more emerging scenarios require different cloud servers to cooperate to the aim of sharing information and/or performing distributed computations. This sharing process can be clearly selective, meaning that different servers may have different access privileges. Recently, a significant amount of research has addressed the problem of processing distributed queries under protection requirements (e.g., [2, 15, 25]). Some proposals are based on the concept of access pattern, a profile associated with each relation/view [15]. For each attribute of the relation/view, the access pattern includes a value that may be either *i* for input or *o* for output. When accessing a relation, the values for all *i* attributes must be supplied to obtain the corresponding values of *o* attributes. Sovereign joins [2] are an alternative solution for securely processing joins. This solution is based on a secure coprocessor, which is involved in query execution, and exploits cryptography. Other approaches propose an authorization model to regulate the view that each server can have on the data, ensuring that query computation exposes to each server only the data that the server can view [25]. The idea is that a relation (base or resulting from the evaluation of a query) can be released to a server whenever the information it carries (either directly or indirectly when the relation has been obtained as the result of a query) is visible from the receiving party. The proposed authorization model operates at the schema level and supports the definition of generic view patterns, thus nicely meeting both expressiveness and simplicity requirements.

Figure 2 summarizes the main categories of security and privacy issues discussed above (gray boxes) along with some of the corresponding solutions (white boxes).

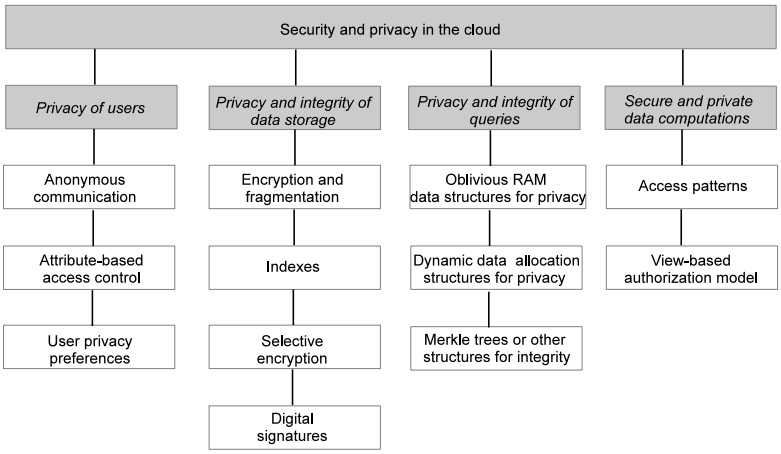


Fig. 2 Summary of security and privacy issues and corresponding solutions

Note that this classification does not aim to be complete but only to provide a quick overview of the solutions mentioned.

3 Privacy of data storage

The problem of protecting outsourced data while enjoying effective and efficient data management and retrieval operations has attracted the attention of many researches, and several investigations have been carried out. The problem is quite complex and involves several aspects, including basic techniques for protecting data at rest (Section 3.1), techniques for efficiently accessing encrypted data without compromising their confidentiality (Section 3.2), and data-centric techniques for supporting selective access to the outsourced data without relying on the data owner and/or on the honest-but-curious server storing the data (Section 3.3). We now describe more in details these aspects.

3.1 Encryption and fragmentation

The problem of protecting the confidentiality of outsourced data has been one of the first issues investigated in the data outsourcing and cloud scenarios. In fact, the risk that unauthorized parties (or even the external server itself) can access sensitive information is one of the main factors for which users (and not only) are often reluctant to adopt the cloud for storing their data. The solutions proposed to protect

PATIENTS

	SSN	Name	YoB	Job	Disease
t_1	123456789	Alice	1980	Clerk	Asthma
t_2	234567891	Bob	1980	Doctor	Asthma
t_3	345678912	Carol	1970	Nurse	Asthma
t_4	456789123	David	1970	Lawyer	Bronchitis
t_5	567891234	Eva	1970	Doctor	Bronchitis
t_6	678912345	Frank	1960	Doctor	Gastritis
t_7	789123456	Gary	1960	Teacher	Gastritis
t_8	891234567	Hilary	1960	Nurse	Diabetes

(a)

$c_0 = \{\texttt{SSN}\}$
$c_1 = \{\texttt{Name}, \texttt{Disease}\}$
$c_2 = \{\texttt{Name}, \texttt{Job}\}$
$c_3 = \{\texttt{Job}, \texttt{Disease}\}$

(b)

Fig. 3 An example of plaintext relation (a) and of a set of confidentiality constraints over it (b)

data confidentiality are based on *encryption* and *fragmentation*, which can be used either singularly or in combination.

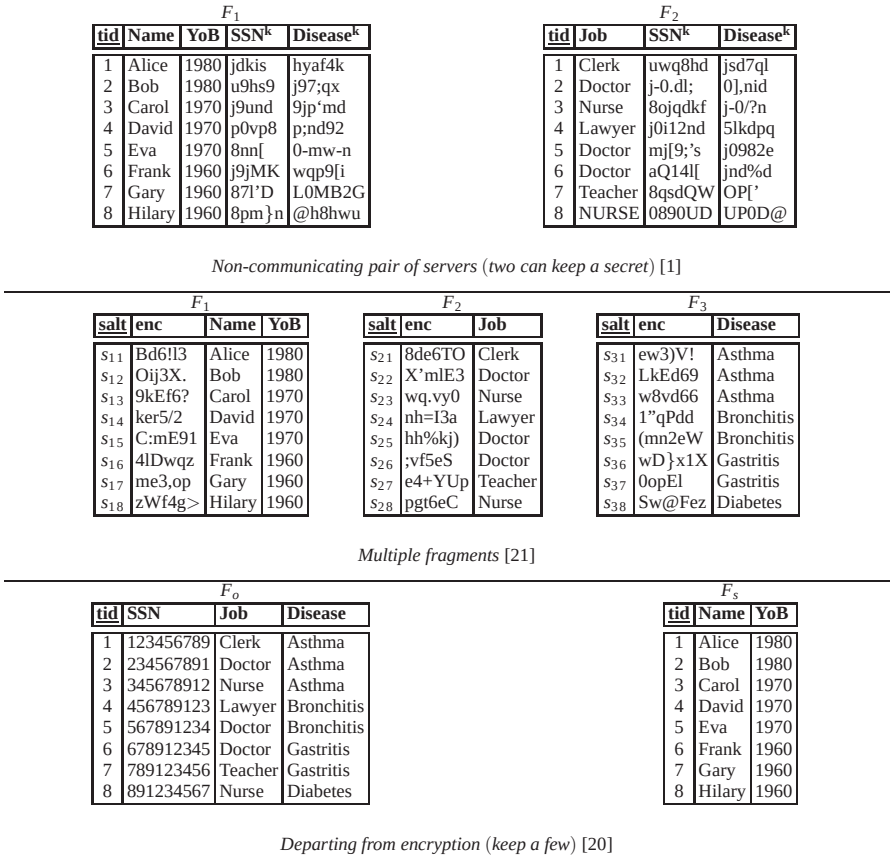
Encryption consists in wrapping a protective layer of encryption around data before storing them at an external server (e.g., [17, 34, 37, 44]). Since the encryption key is known only to the data owner and to authorized users, this technique protects the data against both external (malicious) parties, and the server itself. While effective, this approach is based on the conservative assumption that all the outsourced data are equally sensitive and must therefore be protected. However, as first observed in [1, 20, 21], often data are not sensitive per se but what is sensitive is their association with other data. As an example, the list of the names of hospitalized patients and the list of diseases cured in a hospital are not sensitive. On the contrary, the association of patients' names with the illness they suffer from is highly sensitive and should therefore be kept confidential. Data confidentiality can then be achieved by properly protecting sensitive associations. Given a relation r over relation schema $R(a_1, \ldots, a_n)$, both sensitive attribute values and sensitive associations among them can be modeled through *confidentiality constraints* [1]. A confidentiality constraint c over R is a subset of the attributes in R (i.e., $c \subseteq R$), modeling a sensitive association on the values of the attributes in c. Constraint c states that, for each tuple t in r: *i)* value $t[a]$ is considered sensitive per se, if c is a singleton constraint (i.e., $c=\{a\}$); *ii)* the joint visibility of the values of the attributes in c is considered sensitive, if c is an association constraint (i.e., $c = \{a_i, \ldots, a_j\}$). For instance, Figure 3(b) illustrates a set of confidentiality constraints over relation PATIENTS in Figure 3(a). Singleton constraint c_0 states that the list of Social Security Numbers is considered sensitive per se. The remaining association constraints state that the association of: patients' names with the disease they suffer from (c_1), patients' names with their job (c_2), and patients' job with their disease (c_3) are considered sensitive, respectively.

Given a relation r and a set C of confidentiality constraints over it, the goal is to combine fragmentation and encryption techniques to guarantee that sensitive values and sensitive associations are properly obfuscated. Intuitively, singleton constraints are enforced by encrypting the attribute values before outsourcing or by not outsourcing the attribute values at all. Association constraints are enforced by partitioning the attributes in R in different subsets (*fragments*), or by not releasing (in clear form) at least one of the attributes in the constraint. A fragmentation correctly

enforces the confidentiality constraints if no fragment stored at the external server represents all the attributes in a constraint in clear form, and fragments cannot be joined by unauthorized users.

The approaches that rely on fragmentation and encryption for enforcing confidentiality constraints differ in how they guarantee that fragments cannot be joined, and in how they protect attribute values considered sensitive per se. Based on these differences, existing techniques can be classified as follows.

- *Non-communicating pair of servers* [1]. The data owner partitions relation R in two fragments, F_1 and F_2, stored at two non-communicating servers. Those attributes that cannot be stored at any of the two servers without violating confidentiality constraints are encoded and the result is stored at the two servers (e.g., the attribute values are encrypted via one-time-pad, and the result of encryption is stored at one server, while the key is stored at the other one). Only users who can access both the versions of an encoded attribute can reconstruct its plaintext values. Figure 4 illustrates an example of fragmentation for relation PATIENTS in Figure 3(a) that satisfies the confidentiality constraints in Figure 3(b). It is composed of fragments F_1={tid, Name, YoB, SSN^k, Disease^k} and F_2={tid, Job, SSN^k, Disease^k}. Attribute tid is a tuple identifier introduced in the two fragments to permit authorized users to correctly join F_1 and F_2 to reconstruct the original content of relation PATIENTS. Attributes SSN^k and Disease^k represent the encoded version of attributes SSN and Disease, respectively.
- *Multiple fragments* [21]. The data owner partitions relation R in an arbitrary set of fragments, $\{F_1,\ldots,F_m\}$, possibly stored at the same server. Fragments are disjoint, meaning that no attribute is represented in clear form in more than one fragment. All the attributes in R that are not represented in clear form in a fragment are however represented in encrypted form in the fragment (i.e., each fragment is complete). Figure 4 illustrates an example of fragmentation for relation PATIENTS in Figure 3(a) that satisfies the confidentiality constraints in Figure 3(b). It is composed of three fragments: F_1={salt, enc, Name, YoB}, F_2={salt, enc, Job}, and F_3={salt, enc, Disease}. Attribute salt is a randomly chosen value, different for each tuple in each fragment. Attribute enc is the result of the encryption of the attributes in the original relation that are not represented in clear form in the fragment, concatenated with salt. For readability, in all our examples tuples in fragments are in the same order as in the original relation, even if the order in which tuples are stored in fragments is independent from the order in which they appear in the original relation. Note that the possibility of using an arbitrary number of fragments has the advantage that all attributes that are not involved in singleton constraints can be represented in clear form in a fragment (in the worst case, we can have a fragment for each attribute), as it is visible from the example above.
- *Departing from encryption* [20]. The data owner partitions relation R in two fragments, F_o and F_s, and locally stores one of them (F_o), while the other is outsourced to an external server (F_s). Since only authorized users can access F_o, neither the server nor unauthorized users can join F_o and F_s to possibly reconstruct sensitive associations. Note that fragment F_o can both include attributes



F_1

tid	Name	YoB	SSN^k	Disease^k
1	Alice	1980	jdkis	hyaf4k
2	Bob	1980	u9hs9	j97;qx
3	Carol	1970	j9und	9jp‘md
4	David	1970	p0vp8	p;nd92
5	Eva	1970	8nn[	0-mw-n
6	Frank	1960	j9jMK	wqp9[i
7	Gary	1960	87l’D	L0MB2G
8	Hilary	1960	8pm}n	@h8hwu

F_2

tid	Job	SSN^k	Disease^k
1	Clerk	uwq8hd	jsd7ql
2	Doctor	j-0.dl;	0],nid
3	Nurse	8ojqdkf	j-0/?n
4	Lawyer	j0i12nd	5lkdpq
5	Doctor	mj[9;’s	j0982e
6	Doctor	aQ14l[	jnd%d
7	Teacher	8qsdQW	OP[’
8	NURSE	0890UD	UP0D@

Non-communicating pair of servers (*two can keep a secret*) [1]

F_1

salt	enc	Name	YoB
s_{11}	Bd6!l3	Alice	1980
s_{12}	Oij3X.	Bob	1980
s_{13}	9kEf6?	Carol	1970
s_{14}	ker5/2	David	1970
s_{15}	C:mE91	Eva	1970
s_{16}	4lDwqz	Frank	1960
s_{17}	me3,op	Gary	1960
s_{18}	zWf4g>	Hilary	1960

F_2

salt	enc	Job
s_{21}	8de6TO	Clerk
s_{22}	X’mlE3	Doctor
s_{23}	wq.vy0	Nurse
s_{24}	nh=I3a	Lawyer
s_{25}	hh%kj)	Doctor
s_{26}	;vf5eS	Doctor
s_{27}	e4+YUp	Teacher
s_{28}	pgt6eC	Nurse

F_3

salt	enc	Disease
s_{31}	ew3)V!	Asthma
s_{32}	LkEd69	Asthma
s_{33}	w8vd66	Asthma
s_{34}	1”qPdd	Bronchitis
s_{35}	(mn2eW	Bronchitis
s_{36}	wD}x1X	Gastritis
s_{37}	0opEl	Gastritis
s_{38}	Sw@Fez	Diabetes

Multiple fragments [21]

F_o

tid	SSN	Job	Disease
1	123456789	Clerk	Asthma
2	234567891	Doctor	Asthma
3	345678912	Nurse	Asthma
4	456789123	Lawyer	Bronchitis
5	567891234	Doctor	Bronchitis
6	678912345	Doctor	Gastritis
7	789123456	Teacher	Gastritis
8	891234567	Nurse	Diabetes

F_s

tid	Name	YoB
1	Alice	1980
2	Bob	1980
3	Carol	1970
4	David	1970
5	Eva	1970
6	Frank	1960
7	Gary	1960
8	Hilary	1960

Departing from encryption (*keep a few*) [20]

Fig. 4 An example of fragmentation of relation PATIENTS in Figure 3(a) according to the non-communication pair of servers, multiple fragments, and departing from encryption scenarios

considered sensitive per se and sensitive associations. This solution completely departs from encryption, but it requires the data owner to locally store a portion of her data and to cooperate with the external server in query evaluation. Figure 4 illustrates an example of fragmentation for relation PATIENTS in Figure 3(a) that satisfies the confidentiality constraints in Figure 3(b). It is composed of fragment F_o={tid, SSN, Job, Disease} stored at the data owner side, and fragment F_s={tid, Name, YoB} stored at the external server side.

Encryption, fragmentation, and their combinations are powerful mechanisms for protecting data confidentiality. However, there are still several open issues that need to be further investigated. In fact, fragmentation and encryption break associations among attribute values that could be considered of interest for final recipients, thus compromising the utility of released data. Alternative solutions that protect data while preserving a certain utility are therefore needed [24]. Also, confidentiality

PATIENTS^k

tid	enc	I_n	I_y	I_j	I_d
1	T8/lO?	π	α	δ	η
2	1wfTg<	π	α	ε	θ
3	vFe!d2	ρ	β	δ	ω
4	f3iJ:y	ρ	β	ζ	κ
5	;x0d9D	σ	β	ε	λ
6	kO6i)G	σ	γ	ε	μ
7	u2eW[b	τ	γ	ζ	ν
8	vY7'.1	τ	γ	δ	ξ

Fig. 5 An example of encrypted and indexed version of relation PATIENTS in Figure 3(a)

constraints are defined over relation schemas, while they could be extended to operate at the instance level (i.e., at the attribute values level). We also observe that encryption and fragmentation work under the assumption that the data collection never changes. Techniques supporting updates to the outsourced data collection without compromising confidentiality still need to be designed.

3.2 Indexes

The adoption of encryption for protecting data confidentiality makes query execution difficult. In fact, confidentiality demands that data decryption must be possible only at the user side. Solutions have been then developed to enable cloud servers to execute queries directly on encrypted data. These solutions complement the outsourced relation with a set of *indexes*, which are metadata information built on the plaintext values of the attributes [44]. Formally, a relation r, defined over schema $R(a_1,\ldots,a_n)$, is represented at the server side through an encrypted relation r^k over schema $R^k(\underline{\texttt{tid}}, \texttt{enc}, I_{i_1},\ldots,I_{i_j})$. Attribute `tid` is a numerical attribute added to the original relation and acting as a primary key. Attribute `enc` represents the encrypted tuple. Attribute I_{i_l}, $l = 1,\ldots,j$, is the index defined over attribute a_{i_l} in R. Each tuple t in r is represented by an encrypted tuple t^k in r^k where $t^k[\texttt{enc}]{=}E_k(t)$, with E a symmetric encryption function with key k, and $t^k[I_{i_l}]{=}\iota(t[a_{i_l}])$, with ι an index function defined over D_{i_l}. Note that R^k has an index only for those attributes in R on which conditions need to be evaluated. Figure 5 illustrates an example of encrypted and indexed version of relation PATIENTS in Figure 3(a), with indexes over attributes `Name` (I_n), `YoB` (I_y), `Job` (I_j), and `Disease` (I_d).

Different indexing techniques have been proposed in the literature to support different kinds of conditions. Most of these indexing techniques can be classified in the following three classes, depending on how the corresponding index function ι maps the original values to the corresponding index values.

- *Direct index*. Index function ι maps each plaintext value to a different index value and vice versa. An example of direct index is represented by *encryption-based indexes* (e.g., [22]). For each tuple $t{\in}r$, the value of index I, defined over attribute

a, is computed as $\iota(t[a])=E_k(t[a])$. For instance, index I_y in relation PATIENTS^k in Figure 5 represents an example of direct index over attribute `YoB` of relation PATIENTS in Figure 3(a).

- *Bucket-based index*. Index function ι maps different plaintext values to the same index value, generating collisions. Each plaintext value is however mapped to only one index value. An example of bucket-based index is represented by *partition-based indexes*, which partition the domain D of attribute a into non-overlapping subsets of contiguous values, and associate a label with each partition (e.g., [37]). For each tuple $t \in r$, the value of index I, defined over attribute a, corresponds to the label of the unique partition to which value $t[a]$ belongs. For instance, index I_n in relation PATIENTS^k in Figure 5 represents an example of partition-based index over attribute `Name` of relation PATIENTS in Figure 3(a). The domain of attribute `Name` has been partitioned in four intervals depending on the initial of the name, with labels: π for names with initial in the range [A,B], ρ for names with initial in the range [C,D], σ for names with initial in the range [E,F], and τ for names with initial in the range [G,H]. Another example of bucket-based index is represented by the *hash-based indexes* (e.g., [17]). For each tuple $t \in r$, the value of index I, defined over attribute a, is computed as $\iota(t[a])=h(t[a])$, where h is a secure hash function that generates collisions. For instance, index I_j in relation PATIENTS^k in Figure 5 represents an example of hash-based index over attribute `Job` of relation PATIENTS in Figure 3(a). The hash function adopted generates collisions and, in particular, is defined as follows: h(Clerk)=h(Nurse)=δ, h(Doctor)=ε, and h(Lawyer)=h(Teacher)=ζ.
- *Flattened index*. Index function ι maps each plaintext value to a set of index values to guarantee that all index values have the same number of occurrences (flattening). Each index value represents one plaintext value only. The index can be obtained by applying an encryption function to the plaintext values of the attribute and a post processing that flattens the distribution of the index values (e.g., [45]). For instance, index I_d in relation PATIENTS^k in Figure 5 represents an example of flattened index over attribute `Disease` of relation PATIENTS in Figure 3(a), where each index value has exactly one occurrence.

These indexing techniques support the partial evaluation at the server-side of SQL queries. Given a query q, it is translated into a query q_s executed at the server side on the encrypted relation, and a query q_c executed at the client side on the decrypted result of q_s. Query q_c includes all conditions that cannot be evaluated by the server and aims at eventually discarding all *spurious tuples* returned by q_s, that is, all tuples that do not satisfy the original query submitted by the user. The translation of query q into query q_s and q_c depends both on the kind of indexes defined for the attributes involved in the query and on the kind of query. As an example, consider query q = "SELECT *Att* FROM R WHERE *Cond*", where $Att \subseteq R$ and *Cond* is a set of equality conditions of the form $a=v$, with $a \in R$ and v a constant value in the domain D of a. Each equality condition $a=v$ is translated into an equivalent condition I IN $\iota(v)$, with I the index defined over a and ι the corresponding index function. Query q is then translated into query q_s = "SELECT `enc` FROM R^k WHERE $Cond^k$", where $Cond^k$ includes, for each equality condition $a=v$, the equivalent con-

dition I IN $\iota(v)$. The client will decrypt the result of q_s computed by the server, and will execute query q_c that eliminates spurious tuples, evaluates conditions that cannot be performed at the server side, and projects only the attributes in *Att* to obtain the result of q. For instance, query q = SELECT `Name` FROM PATIENTS WHERE `Job`='Nurse' AND `Disease`='Asthma' is translated into query q_s = SELECT `enc` FROM PATIENTS^k WHERE I_j=δ AND $I_d \in \{\eta,\theta,\omega\}$, which returns the first and third tuples in Figure 5. The client then filters spurious tuples from the result of q_s by evaluating query q_c = SELECT `Name` FROM $D_k(Res^k)$ WHERE `Job`='Nurse', where Res^k is the encrypted result returned by the server and D the symmetric decryption function with key k. Query q_c returns the value of attribute `Name` of tuple t_1 in Figure 3(a), which corresponds to the result of the original query q formulated by the user.

Indexing techniques specifically aimed at supporting the efficient evaluation of range conditions are based on order preserving encryption schemas (e.g., [3, 45]). Indexes that support aggregate functions and the basic arithmetic operators (i.e., $+,-,\times$) rely on homomorphic encryption techniques (e.g., [32, 36]). Additional indexing techniques, which cannot be classified as mentioned above, are based, for example, on the definition of data structures (e.g., $B+$-tree) coupled with the encrypted relation and stored at the server [22].

The definition of indexes over outsourced relations must balance precision in query evaluation and privacy of the data [17]. In fact, more precise indexes provide more efficient query execution, at the price of a greater exposure to possible privacy violations. Also, the number of indexes complementing an outsourced relation should be carefully tuned, since each additional index may cause a rapid growth to the risk of privacy violations.

3.3 Selective encryption

In many real-world systems, different users may have different privileges on the outsourced data. Traditional access control architectures are based on the presence of a trusted component, called *reference monitor*, that is in charge of enforcing the access control policy defined by the data owner. In a cloud scenario, however, neither the data owner (for efficiency reasons) nor the cloud server storing the data (for privacy reasons) can enforce the access control policy. An interesting solution addressing this issue consists in adopting *selective encryption* [23], meaning that different keys are used for encrypting different data. The encryption keys are then (directly or indirectly) released only to the users authorized to access the corresponding data. The idea of using different keys for enforcing access control is not new and has been first introduced in other contexts. For instance, in [42] the authors propose to store encrypted XML documents on (potentially insecure and vulnerable) Web servers. The decisions about access rights to different portions of an XML document can be made by the document creator and are immediately applied to the XML document by using different encryption keys for different portions of the same

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
A	1	0	0	1	0	1	1	0
B	0	1	0	1	1	1	1	0
C	0	0	0	1	0	1	1	1
D	0	1	1	1	1	1	0	0
E	0	1	0	0	1	1	1	0

Fig. 6 An example of access matrix regulating access to relation PATIENTS in Figure 3(a)

XML document. To enforce access restrictions, users then obtain only the keys associated with the portions of XML documents for which they have an access right. Other proposals put forward the idea of using hierarchical-based access control in the context of distributed environments and broadcast pay tv content (e.g., [12, 49]). In the remainder of this section, we describe the main characteristics of the selective encryption approach in [23], specifically designed for the cloud scenario.

Given a set U of users and a relation r, the authorization policy regulating access to tuples in r is represented by an access matrix M, with a row for each user $u \in U$ and a column for each tuple $t \in r$. Cell $M[u,t]$ is equal to 1 (0, respectively), if user u can (cannot, respectively) access tuple t. For each tuple t, $acl(t)$ denotes the set of users who can access it (i.e., its access control list). For instance, Figure 6 illustrates an example of access matrix regulating access to the tuples of relation PATIENTS in Figure 3(a) by a set $U=\{A,B,C,D,E\}$ of users.

The authorization policy defined by the data owner is translated into an *equivalent encryption policy*. The encryption policy regulates keys used to encrypt tuples as well as key distribution to users and must be equivalent to the access control policy defined by the data owner, that is, each user can decrypt all and only the tuples she is authorized to access.

The translation of an authorization policy into an equivalent encryption policy is driven by two requirements: *i)* each user must manage at most one key, and *ii)* each tuple must be encrypted at most once (i.e., no replication). To satisfy these two desiderata, the approach in [23] adopts a *key derivation technique* based on public tokens, which permit to compute the value of an encryption key starting from the knowledge of another key and a piece of publicly available information [10]. Each key k_i is associated with a public label l_i and, given keys k_i and k_j, token $token_{i,j}$ is computed as $k_j \oplus h(k_i,l_j)$, with $\oplus$ the bitwise xor operator, and h a deterministic cryptographic function. Token $token_{i,j}$ permits to derive key k_j from k_i and public label l_j. Key derivation techniques are based on the definition of a *key derivation graph*, specifying which keys can be derived from other keys. A key derivation graph is a directed acyclic graph whose vertices represent keys, and whose edges represent tokens. The existence of a path from key k_i to key k_j in the key derivation graph denotes the fact that k_j can be (directly or indirectly, via a chain of tokens) derived from k_i. A key derivation graph correctly enforces an authorization policy M if each user $u_i \in U$ can derive, starting from the key she knows, the keys used to encrypt all and only the tuples $t_j \in r$ that she can access (i.e., with $M[u_i,t_j]=1$). To define such a graph, the idea is to exploit the set containment relationship $\subseteq$ over

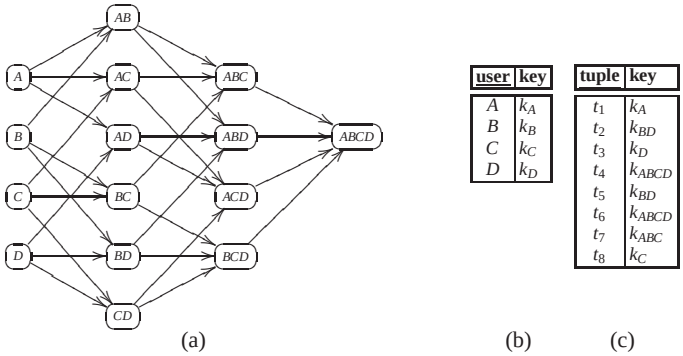
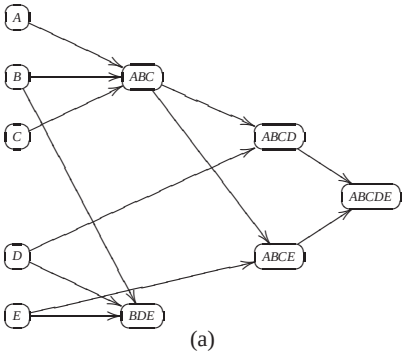


Fig. 7 An example of encryption policy equivalent to the access control policy in Figure 6, considering the subset $\{A,B,C,D\}$ of users

U. A key derivation graph induced by $\subseteq$ over U has a vertex for each subset of users in 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 correct enforcement of the policy is guaranteed if each user knows the key of the vertex representing herself in the graph, and each tuple is encrypted with the key of the vertex representing its acl. For instance, consider the portion of the access matrix in Figure 6 defined for the subset $\{A,B,C,D\}$ of users. The encryption policy in Figure 7 is equivalent to the access control policy represented by the first four rows in Figure 6. For readability, each vertex in the graph of Figure 7 is labeled with the set of users it represents. As an example, user A can decrypt tuples t_1, t_4, t_6, and t_7 since she can derive, starting from vertex labeled A, the keys with which these tuples are encrypted.

Although effective for enforcing the authorization policy, the solution above defines more keys and tokens than necessary. Since the number of tokens in the system influences the access time, the proposal in [23] reduces the number of tokens by removing from the key derivation graph the vertices and edges that are not necessary to enforce M. The problem of minimizing the number of edges in a key derivation graph is however NP-hard. In [23] the authors propose an heuristic approach, which has been proved to obtain good results, based on two observations: *i)* the vertices needed for correctly enforcing an authorization policy are those representing singleton sets of users and the acls of tuples in r; *ii)* when two or more vertices have more than two common direct ancestors, the insertion of a vertex representing the set of users corresponding to these ancestors reduces the total number of tokens. Figure 8(a) illustrates an example of key derivation graph obtained adopting the approach in [23] over the access matrix in Figure 6. As it is visible from the figure, the graph includes a vertex for each user and for each acl of a tuple in the system. It also includes an additional vertex (i.e., *ABC*), introduced to limit the number of tokens in the system. Clearly, the encryption policy in Figure 8 is more convenient than the one in Figure 7, as it reduces both the number of keys and the number of tokens in the system, while managing an additional user.



(a)

user	key
A	k_A
B	k_B
C	k_C
D	k_D
E	k_E

(b)

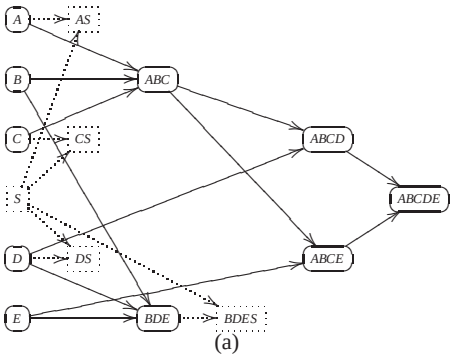
tuple	key
t_1	k_A
t_2	k_{BDE}
t_3	k_D
t_4	k_{ABCD}
t_5	k_{BDE}
t_6	k_{ABCDE}
t_7	k_{ABCE}
t_8	k_C

(c)

Fig. 8 An example of encryption policy equivalent to the access control policy in Figure 6

Since the keys used to encrypt tuples depend on their access control lists, whenever the authorization policy changes, the tuples involved in the policy update may need to be re-encrypted to guarantee the equivalence of the encryption policy. For instance, assume that user E is revoked the privilege to read tuple t_6. Such a tuple should be first decrypted using key k_{ABCDE}, and then encrypted using key k_{ABCD}. However, re-encryption requires the direct involvement of the data owner and can be computationally expensive. The number of re-encryption operations are therefore minimized by adopting two layers of encryption that allow the server to manage policy update operations [23]. The *Base Encryption Layer* (BEL) is applied by the data owner before transmitting the relation to the server and consists in encrypting the tuples according to the authorization policy existing at initialization time. The *Surface Encryption Layer* (SEL) is performed by the server over the tuples already encrypted by the data owner. It enforces the dynamic changes over the policy. The basic idea consists in over-encrypting the tuples so that a user can access a tuple only if she knows or can derive the key used for encrypting the tuples at both levels.

The solution in [23] enforces read privileges only and has been complemented with another technique that allows the management of write operations [27]. This work associates each tuple with a *write tag*. The write tag is a random value chosen by the data owner independently from the tuple content, and is encrypted with a key known only to users who can modify the tuple and to the external server. The server will then enforce a write operation on a tuple only if the requesting user proves to know the write tag of the tuple. The proposal in [27] extends the key derivation graph with a key for the server and the keys necessary for protecting write tags. For instance, consider the read privileges in Figure 6 over relation PATIENTS in Figure 3(a), and assume that: tuples t_1, t_4, and t_7 can be modified by user A only; tuples t_2 and t_6 can be modified by B, D, and E; tuples t_3 and t_5 can be modified by D; and tuple t_8 can be modified by C. Figure 9 illustrates the encryption policy in Figure 8, extended to properly enforce write privileges. In the figure, we denote the external server as S.



(a)

user	key
A	k_A
B	k_B
C	k_V
D	k_D
E	k_E

(b)

tuple	read key	write key
t_1	k_A	k_{AS}
t_2	k_{BDE}	k_{BDES}
t_3	k_D	k_{DS}
t_4	k_{ABCD}	k_{AS}
t_5	k_{BDE}	k_{DS}
t_6	k_{ABCDE}	k_{BDES}
t_7	k_{ABCE}	k_{AS}
t_8	k_C	k_{CS}

(c)

Fig. 9 Encryption policy in Figure 8, extended to enforce write authorizations

Open issues that still need to be addressed are related to the expressive power of the supported access control policy, especially considering the ever-increasing bring-your-own-device (BYOD) trend. In fact, it would be interesting to develop solutions that will allow the specification of fine-grained restrictions, based on the users' context and on the specific device adopted for accessing data.

4 Indexes and fragmentation

The fragmentation works illustrated in Section 3.1 permit to delegate to the server the evaluation of any condition over attributes appearing plaintext in a fragment. However, the client still needs to evaluate those queries that operate on encrypted attributes, or that involve attributes that are not represented in plaintext in the same fragment. For instance, consider the fragmentation in Figure 4 of relation PATIENTS in Figure 3(a). Query q = SELECT `Name` FROM PATIENTS WHERE `YoB`=1980 AND `Disease`='Asthma' cannot be evaluated by the server, since attributes `YoB` and `Disease` do not appear in the clear in the same fragment and the server can neither decrypt attribute `enc` nor join F_1 and F_3. Hence, one of the two conditions in q must be evaluated by the client. To mitigate the client's overhead in query evaluation, fragments can be complemented with indexes over encrypted attributes. Figure 10 illustrates three versions of fragment F_1 in Figure 4, complemented with index I_d over attribute `Disease`, which has been computed using each of the three kinds of indexes illustrated in Section 3.2. The presence of indexes in a fragment could however cause unintended leakage of sensitive information [28]. The exposure to leakage varies depending on the knowledge that a curious observer (e.g., the external server) can exploit and the kind of indexes. In particular, the following two kinds of knowledge can be exploited for breaching data confidentiality.

F_1

salt	enc	Name	YoB	I_d
s_{11}	Bd6!l3	Alice	1980	α
s_{12}	Oij3X.	Bob	1980	α
s_{13}	9kEf6?	Carol	1970	α
s_{14}	ker5/2	David	1970	β
s_{15}	C:mE91	Eva	1970	β
s_{16}	4lDwqz	Frank	1960	γ
s_{17}	me3,op	Gary	1960	γ
s_{18}	zWf4g>	Hilary	1960	δ

(a)

F_1

salt	enc	Name	YoB	I_d
s_{11}	Bd6!l3	Alice	1980	ε
s_{12}	Oij3X.	Bob	1980	ε
s_{13}	9kEf6?	Carol	1970	ε
s_{14}	ker5/2	David	1970	η
s_{15}	C:mE91	Eva	1970	η
s_{16}	4lDwqz	Frank	1960	θ
s_{17}	me3,op	Gary	1960	θ
s_{18}	zWf4g>	Hilary	1960	ε

(b)

F_1

salt	enc	Name	YoB	I_d
s_{11}	Bd6!l3	Alice	1980	κ
s_{12}	Oij3X.	Bob	1980	λ
s_{13}	9kEf6?	Carol	1970	μ
s_{14}	ker5/2	David	1970	ν
s_{15}	C:mE91	Eva	1970	ξ
s_{16}	4lDwqz	Frank	1960	π
s_{17}	me3,op	Gary	1960	ρ
s_{18}	zWf4g>	Hilary	1960	σ

(c)

Fig. 10 Fragment F_1 in Figure 4 complemented with a direct index (a), a bucket-based index (b), and a flattened index (c) over attribute `Disease`

Disease
Asthma
Asthma
Asthma
Bronchitis
Bronchitis
Gastritis
Gastritis
Diabetes

(a)

Name	Disease
Alice	Asthma

(b)

Fig. 11 An example of vertical (a) and horizontal (b) knowledge by an observer

- *Vertical knowledge* is the knowledge of the projection of attribute a over relation r, and is due to the presence of attribute a in the clear in one fragment and indexed in other fragments. Vertical knowledge does not require any additional external information for an observer since, apart from the case where the attribute appears in a singleton constraint, it refers to information immediately present in other accessible fragments. For instance, fragment F_3 in Figure 4 makes visible the plaintext values (and their number of occurrences) of attribute `Disease` (see Figure 11(a)).
- *Horizontal knowledge* is the knowledge of the presence of a tuple t (or a set thereof) in r, and is due to external knowledge by an observer. For instance, an observer may know that Alice suffers from Asthma (see Figure 11(b)).

Let us now examine the exposure risk of indexed fragments under the assumptions of horizontal and vertical knowledge and of the presence of indexes belonging to the three categories discussed in Section 3.2 [28].

- *Direct index*. Index function ι preserves the frequency distribution of plaintext values, which can be exploited to reconstruct the value-index association by an observer with vertical and/or horizontal knowledge. Vertical knowledge permits to precisely reconstruct the value-index association for values characterized by a unique number of occurrences (outliers). For instance, consider the indexed fragment in Figure 10(a) and the vertical knowledge in Figure 11(a). It is immediate to see that ι(Asthma)=α and ι(Diabetes)=δ since these are the only plaintext

and index values with 3 occurrences and 1 occurrence, respectively. Hence, an observer can infer that Alice, Bob, and Carol have Asthma and Hilary has Diabetes. Horizontal knowledge permits to precisely reconstruct the value-index association for the plaintext value $v{=}t[a]$ known by the observer, exposing all the tuples in r with value v for attribute a. For instance, in the example above, knowing that Alice suffers from Asthma permits an observer to infer that ι(Asthma)=α and then that also Bob and Carol suffer from the same illness.

- *Bucket-based index*. Index function ι does not preserve the frequency distribution of plaintext values. However, the index value corresponding to plaintext value v will have a frequency equal to or higher than (in case of collisions) the frequency of v. Values with a high number of occurrences (outliers) are then still exposed. Vertical knowledge permits to identify the index values associated with frequent plaintext values, and then to reconstruct the value-index association for such values with a known probability of error. For instance, consider the indexed fragment in Figure 10(b) and the vertical knowledge in Figure 11(a). Clearly, ι(Asthma)=ε since this is the only index value with at least 3 occurrences. Also, ι(Diabetes)=ε since Diabetes is the only plaintext value with 1 occurrence. An observer can then infer that 3 patients among Alice, Bob, Carol, and Hilary has Asthma (each with probability 0.75) and 1 has Diabetes (each with probability 0.25). Horizontal knowledge permits to identify the index value representing the known plaintext value $v{=}t[a]$. This index value may however correspond also to other plaintext values, limiting the observer's ability to precisely reconstruct value-index associations. For instance, in the example above, knowing that Alice suffers from Asthma permits an observer to infer that ι(Asthma)=ε. However, nothing can be said about Bob, Carol, and Hilary since ε could also represent other plaintext values (different from Asthma). By combining horizontal with vertical knowledge, however, she can infer that 2 among Bob, Carol, and Hilary suffer from Asthma (each with probability 0.66) and 1 suffers from Diabetes (each with probability 0.33).
- *Flattened index*. Index function ι flattens the frequency distribution of index values. Vertical knowledge does not help in establishing correspondences between plaintext values and index values. Horizontal knowledge permits to identify one of the index values representing the known plaintext value $v{=}t[a]$, exposing only the tuples associated with this index value (in contrast to the possibly larger set of tuples with value v for a). For instance, consider the indexed fragment in Figure 10(c) and the horizontal knowledge in Figure 11(b). An observer can only learn that ι(Asthma)=κ. However, no other association is exposed, because κ has only one occurrence in F_1 (although Asthma has frequency 3 in F_3).

An index function ι that flattens the frequency distribution of index values and that generates collisions provides protection against both horizontal and vertical knowledge. In fact, as illustrated above, inference attacks caused by vertical knowledge can be counteracted by flattening the frequency distribution of index values. Inference attacks caused by horizontal knowledge are mitigated by index functions that map different plaintext values to the same index value, generating collisions. For instance, Figure 12 illustrates fragment F_1 in Figure 4 complemented with a

F_1

salt	enc	Name	YoB	I_d
s_{11}	Bd6!l3	Alice	1980	α
s_{12}	Oij3X.	Bob	1980	α
s_{13}	9kEf6?	Carol	1970	δ
s_{14}	ker5/2	David	1970	β
s_{15}	C:mE91	Eva	1970	β
s_{16}	4lDwqz	Frank	1960	γ
s_{17}	me3,op	Gary	1960	γ
s_{18}	zWf4g>	Hilary	1960	δ

Fig. 12 Fragment F_1 in Figure 4 complemented with a flattened index with collisions over attribute `Disease`

flattened index with collisions over attribute `Disease`. This indexed fragment is protected against both vertical and horizontal knowledge in Figure 11. Indeed, vertical knowledge cannot be exploited for frequency-based attacks (all the index values have 2 occurrences). Horizontal knowledge permits to infer that ι(Asthma)=α but, since ι generates collisions, the observer cannot say anything about the disease from which Bob suffers. Although the proposal in [28] is focused on the adoption of one index, the discussion can easily be extended to the case where fragments are complemented with multiple indexes. In fact, flattening and collisions provide adequate protection in different scenarios (e.g., multiple indexes in one fragment, a same attribute indexed in different fragments, two attributes appearing one in plaintext and the other indexed in one fragment and reversed in another fragment).

Although effective to protect data at rest, a flattened index function with collisions has the disadvantage of reducing the performance in query evaluation. In fact, flattening requires to retrieve different index values when searching for one plaintext value, and collisions require a post-processing at the client side to remove spurious tuples in the query result computed by the server. As an example, consider fragment F_1 in Figure 12, condition `Disease`='Asthma' translates into condition I_d IN $\{\alpha,\delta\}$. The evaluation of this condition would however return a tuple with value Diabetes for attribute `Disease` (i.e., tuple t_8), since Asthma and Diabetes are both mapped to value δ. Also, flattened indexes with collisions remain still vulnerable to dynamic observations (i.e., to adversaries who can observe users' queries). In fact, by observing a long enough sequence of queries, an observer can easily infer the index values to which each plaintext value has been mapped, since they always appear together in query conditions. With reference to the example above, every query including condition `Disease`='Asthma' is translated into a query including condition I_d IN $\{\alpha,\delta\}$. An observer can then easily infer that α and δ represent the same plaintext value (Asthma, in our example). The protection against dynamic observations represents an open issue that still needs to be addressed, along with the problem of defining an efficient index function that provides both flattening and collisions.

t	acl(t)
t_1	*A*
t_2	*BDE*
t_3	*D*
t_4	*ABCD*
t_5	*BDE*
t_6	*ABCDE*
t_7	*ABCE*
t_8	*C*

(a)

PATIENTS

	SSN	Name	YoB	Job	Disease
t_1	123456789	Alice	1980	Clerk	Asthma
t_2					
t_3					
t_4	456789123	David	1970	Lawyer	Bronchitis
t_5					
t_6	678912345	Frank	1960	Doctor	Gastritis
t_7	789123456	Gary	1960	Teacher	Gastritis
t_8					

(b)

PATIENTS^k

tid	enc	I_n	I_y	I_j	I_d
1	T8/lO?	π	α	δ	η
2	1wfTg<	π	α	ε	θ
3	vFe!d2	ρ	β	δ	ω
4	f3iJ:y	ρ	β	ζ	κ
5	;x0d9D	σ	β	ε	λ
6	kO6i)G	σ	γ	ε	μ
7	u2eW[b	τ	γ	ζ	ν
8	vY7'.1	τ	γ	δ	ξ

(c)

Fig. 13 Knowledge of user *A* over relation PATIENTS (b) and PATIENTS^k (c)

5 Indexes and selective encryption

Selective encryption approaches illustrated in Section 3.3 enforce access control restrictions over outsourced data by guaranteeing that each user can decrypt all and only the tuples she is authorized to access. However, when data are made selectively available, the combination of selective encryption with indexes used for enabling efficient query execution on encrypted data may open the door to inferences. In fact, users may have visibility of indexes even of tuples they are not allowed to access. Such visibility, together with their ability to view data for which they are authorized, can allow them to possibly infer plaintext values of tuples they should not be able to read. In the following, for clarity in the exposition but without loss of generality, we will refer the discussion to one attribute *a* only.

The knowledge that a user u can exploit for inferences can be summarized as follows: *i)* index function ι used to define index I over attribute a (necessary to translate user' queries into queries that operate at the server side); *ii)* plaintext tuples that the user can access (i.e., t such that $u \in acl(t)$); *iii)* all the encrypted tuples in r^k. For instance, consider relation PATIENTS in Figure 3(a) and the authorization policy in Figure 6 (which is also summarized in Figure 13(a) for the reader's convenience), Figures 13(b) and (c) illustrate the knowledge of user *A* over the plaintext and encrypted relation. Gray cells denote values that *A* is not authorized to read.

The information that a user with this knowledge can infer depends on the kind of index adopted (see Section 3.2), as illustrated in the following [26].

- *Direct index*. Index function ι is a bijective function that maps each plaintext value to one index value (and vice versa). It then exposes all the tuples with the same plaintext value for attribute a of a tuple that the user is authorized to access. For instance, index I_y over attribute YoB in Figure 13(c) has been computed using a direct index function. Since user *A* can access tuple t_1, she knows that $\iota(1980)=\alpha$. She can then infer that t_2[YoB]=1980, even if she is not authorized to access tuple t_2. In a similar way, *A* can also infer that $\iota(1970)=\beta$ and that $\iota(1960)=\gamma$ (i.e., she knows the plaintext value of attribute YoB of each tuple in PATIENTS). The user also knows index function ι. Hence, she can compute the index value $\iota(v)$ associated with each value v in the domain of attribute a,

t	acl(t)
t_1	*A*
t_2	*BDE*
t_3	*D*
t_4	*ABCD*
t_5	*BDE*
t_6	*ABCDE*
t_7	*ABCE*
t_8	*C*

(a)

PATIENTS

	SSN	Name	YoB	Job	Disease
t_1	123456789	Alice	1980	Clerk	Asthma
t_2			*1980*		*Asthma*
t_3			*1970*		*Asthma*
t_4	456789123	David	1970	Lawyer	Bronchitis
t_5			*1970*		*Bronchitis*
t_6	678912345	Frank	1960	Doctor	Gastritis
t_7	789123456	Gary	1960	Teacher	Gastritis
t_8			*1960*		

(b)

PATIENTS^k

tid	enc	I_n	I_y	I_j	I_d
1	T8/lO?	π	α	δ	η
2	1wfTg<	π	α	ε	θ
3	vFe!d2	ρ	β	δ	ω
4	f3iJ:y	ρ	β	ζ	κ
5	;x0d9D	σ	β	ε	λ
6	kO6i)G	σ	γ	ε	μ
7	u2eW[b	τ	γ	ζ	ν
8	vY7'.1	τ	γ	δ	ξ

(c)

Fig. 14 Knowledge inferred by user *A* over relation PATIENTS

and possibly reconstruct the value that attribute a assumes in each tuple t of the outsourced relation, independently from her access privileges over t.

- *Bucket-based index.* Index function ι is a surjective function that maps multiple plaintext values to one index value. The inference risks described for direct indexes are mitigated by collisions. In fact, multiple occurrences of a same index value may correspond to different plaintext values. The user's knowledge of index function ι could however reduce the uncertainty over the value assumed by attribute a in a tuple t that she is not authorized to access. For instance, index I_j over attribute `Job` in Figure 13(c) has been computed using a bucket-based index function. Since user *A* can access tuple t_1, she knows that ι(Clerk)=δ. However, she does not know with certainty whether t_3[`Job`]=Clerk and t_8[`Job`]=Clerk since function ι may generate collisions and map different plaintext values to index value δ.
- *Flattened index.* Index function ι is an injective function that maps a plaintext value to multiple index values, guaranteeing a flat distribution of the number of occurrences of index values. Like direct indexes, flattened indexes expose all the tuples with the same plaintext value for attribute a of a tuple that the user is authorized to access. In fact, when decrypting a tuple t that she can access, the user knows one of the index values representing value v=t[a]. By computing $\iota(v)$, she exactly knows which tuples in r^k have value v for attribute a. For instance, index I_d over attribute `Disease` in Figure 13(c) has been computed using a flattened index function. Since user *A* can access tuple t_1, she knows that ι(Asthma)=η and, since she can compute $\iota(v)$ for any v in the domain of attribute `Disease`, she can compute the set of index values representing Asthma, that is, $\{\eta,\theta,\omega\}$. She can then infer that t_2[`Disease`]=t_3[`Disease`]=Asthma.

Inferences by user *A* over relation PATIENTS are summarized in Figure 14, where light-gray cells represent values, reported in italic, that *A* is not authorized to access but that she can infer from her knowledge.

From the observations above, we note that inference is mainly caused by the presence of the same index value associated with tuples characterized by different authorizations. In [26] the authors proposed a solution, which is focused on direct indexes since they represent the worst case scenario, based on the principle that

PATIENTS^k

tid	enc	I_y
1	T8/lO?	α_A
2	1wfTg<	α_B, α_D, α_E
3	vFe!d2	β_D
4	f3iJ:y	β_A, β_B, β_C, β_D
5	;x0d9D	β_B, β_D, β_E
6	kO6i)G	γ_A, γ_B, γ_C, γ_D, γ_E
7	u2eW[b	γ_A, γ_B, γ_C, γ_E
8	vY7'.1	γ_C

(a)

PATIENTS^k

tid	enc	I_y
1	T8/lO?	α_A
2	1wfTg<	α_B, α_D, α_E
3	vFe!d2	β_D
4	f3iJ:y	β_A, β_B, β_C, β'_D
5	;x0d9D	β'_B, β''_D, β_E
6	kO6i)G	γ_A, γ_B, γ_C, γ_D, γ_E
7	u2eW[b	γ'_A, γ'_B, γ'_C, γ'_E
8	vY7'.1	γ''_C

(b)

Fig. 15 An example of encrypted and indexed version of relation PATIENTS with index I_y over YoB computed using a user-dependent function (a) and a salted user-dependent function (b)

different occurrences of the same index value must be mapped to different index values when they should be visible to different subsets of users. The index value to which $t[a]$ should be mapped therefore depends, not only on value v=$t[a]$, but also on $acl(t)$. To this purpose, each user u has its own index function ι_u, which depends on a private piece of information that she shares with the data owner. Given a tuple t, the data owner computes a different index value $\iota_u(t[a])$ for each $u \in acl(t)$. Each user will then use her index function ι_u to formulate queries to be evaluated by the external server over indexes. For instance, Figure 15(a) illustrates relation PATIENTS^k, where the index over attribute YoB has been computed adopting a user-dependent function. In the figure, for simplicity, we indicate with a sub-script the user whose index function generated the value (i.e., v_u is a value generated by ι_u). Note that $v_{u_i} \neq v_{u_j}$.

Since all the index values associated with a specific plaintext value of attribute a are visible to all the users in the system, the adoption of user-dependent index functions is not sufficient to block all the inferences. In fact, tuples sharing the same value for attribute a that are characterized by different but overlapping acls, called *conflicting tuples*, are exposed to inferences by users who can access at least one of these tuples. For instance, with reference to relation PATIENTS^k in Figure 15(a), user A cannot exploit her knowledge of tuple t_1 to infer the value of t_2[YoB]. However, by observing that β_D appears in tuples t_4 together with β_A, A can infer that β_D represents value 1970 and hence that t_3[YoB]=t_4[YoB]=t_5[YoB]=1970. To block this inference channel, conflicting tuples must be associated with disjoint sets of index values. To impose diversity of indexes, the value computed by index function ι_u is differentiated by applying different randomly generated salts to conflicting tuples. For instance, Figure 15(a) illustrates relation PATIENTS^k, where the index over attribute YoB has been computed adopting a salted user-dependent function. In the figure, we denote salted versions of value v as v' and v''.

While effective, the solution illustrated above presents similar privacy risks to the one described in Section 4. More precisely, this indexing technique remains vulnerable to dynamic observations, since monitoring a sufficient number of queries would permit an observer to reconstruct which (salted) index values represent the same plaintext value. Furthermore, collusion between authorized users and the ex-

ternal server may put data confidentiality at risk. The protection against these threats still remains an open issue.

6 Conclusions

Cloud computing offers a variety of new opportunities to users and companies, and many efforts have been therefore dedicated to the design of cloud-based services, applications, and infrastructures. While appealing, cloud computing however introduces new security and privacy issues. In this chapter, we analyzed the data protection issues, and described approaches for the protection of data confidentiality, and for the efficient and selective access to data. We also illustrated open problems arising from the combined application of such solutions and highlighted possible directions to address them.

Acknowledgements The chapter is based on joint work with Sushil Jajodia and Stefano Paraboschi. This work was supported in part by the Italian Ministry of Research within PRIN 2010-2011 project "GenData 2020" (2010RTFWBH), and by Google under the Google Research Award program.

References

1. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: A distributed architecture for secure database services. In: Proc. of CIDR 2005. Asilomar, CA, USA (January 2005)
2. Agrawal, R., Asonov, D., Kantarcioglu, M., Li, Y.: Sovereign joins. In: Proc. of ICDE 2006. Atlanta, GA, USA (April 2006)
3. Agrawal, R., Kierman, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of SIGMOD 2004. Paris, France (June 2004)
4. Ardagna, C., Camenisch, J., Kohlweiss, M., Leenes, R., Neven, G., Priem, B., Samarati, P., Sommer, D., Verdicchio, M.: Exploiting cryptography for privacy-enhanced access control: A result of the PRIME project. JCS 18(1), 123–160 (2010)
5. Ardagna, C., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Samarati, P.: Minimizing disclosure of private information in credential-based interactions: A graph-based approach. In: Proc. of PASSAT 2010. Minneapolis, MN, USA (August 2010)
6. Ardagna, C., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Samarati, P.: Supporting privacy preferences in credential-based interactions. In: Proc. of WPES 2010. Chicago, IL, USA (October 2010)
7. Ardagna, C., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Samarati, P.: Minimising disclosure of client information in credential-based interactions. IJIPSI 1(2/3), 205–233 (2012)
8. Ardagna, C., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., Samarati, P., Verdicchio, M.: Expressive and deployable access control in open web service applications. IEEE TSC 4(2), 96–109 (April-June 2011)
9. Ardagna, C., Jajodia, S., Samarati, P., Stavrou, A.: Providing users' anonymity in mobile hybrid networks. ACM TOIT (2013)

10. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM TISSEC 12(3), 18:1–18:43 (January 2009)
11. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: On the propagation of faults and their detection in a hardware implementation of the advanced encryption standard. In: Proc. of ASAP 2002. San Jose, CA, USA (July 2002)
12. Blanton, M., Frikken, K.: Efficient multi-dimensional key management in broadcast services. In: Proc. of ESORICS 2010. Athens, Grece (September 2010)
13. Bonatti, P., Samarati, P.: A uniform framework for regulating service access and information release on the Web. JCS 10(3), 241–272 (2002)
14. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Proc. of EUROCRYPT 2003. Warsaw, Poland (May 2003)
15. Calì, A., Martinenghi, D.: Querying data under access limitations. In: Proc. of ICDE 2008. Cancun, Mexico (April 2008)
16. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Proc. of EUROCRYPT 2001. Innsbruck, Austria (May 2001)
17. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC 8(1), 119–152 (February 2005)
18. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of ACM 45(6), 965–981 (April 1998)
19. Cimato, S., Gamassi, M., Piuri, V., Sassi, R., Scotti, F.: Privacy-aware biometrics: Design and implementation of a multimodal verification system. In: Proc. of ACSAC 2008. Anaheim, CA, USA (December 2008)
20. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a few: Outsourcing data while maintaining confidentiality. In: Proc. of ESORICS 2009. Saint Malo, France (September 2009)
21. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM TISSEC 13(3), 22:1–22:33 (July 2010)
22. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of CCS 2003. Washington, DC, USA (October 2003)
23. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM TODS 35(2), 12:1–12:46 (April 2010)
24. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragments and loose associations: Respecting privacy in data publishing. PVLDB 3(1), 1370–1381 (September 2010)
25. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Authorization enforcement in distributed query evaluation. JCS 19(4), 751–794 (2011)
26. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Private data indexes for selective access to outsourced data. In: Proc. of WPES 2011. Chicago, IL, USA (October 2011)
27. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Support for write privileges on outsourced data. In: Proc. of SEC 2012. Heraklion, Crete, Greece (June 2012)
28. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: On information leakage by indexes over data fragments. In: Proc. of PrivDB 2013. Brisbane, Australia (April 2013)
29. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of ICDCS 2011. Minneapolis, MN, USA (June 2011)
30. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Supporting concurrency in private data outsourcing. In: Proc. of ESORICS 2011. Leuven, Belgium (September 2011)

31. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Protecting data in outsourcing scenarios. In: Das, S., Kant, K., Zhang, N. (eds.) Handbook on Securing Cyber-Physical Critical Infrastructure. Morgan Kaufmann (2012)
32. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. of STOC 2009. Bethesda, MA, USA (May 2009)
33. Goodrich, M., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless Oblivious RAM simulation. In: Proc. of SODA 2012. Kyoto, Japan (January 2012)
34. Hacigümüs, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proc. of ICDE 2002. San Jose, CA, USA (February 2002)
35. Hacigümüs, H., Iyer, B., Mehrotra, S.: Ensuring integrity of encrypted databases in database as a service model. In: Proc. of DBSec 2003. Estes Park, CO, USA (August 2003)
36. Hacigümüs, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: Proc. of DASFAA 2004. Jeju Island, Korea (March 2004)
37. Hacigümüs, H., Iyer, B., Mehrotra, S., Li, C.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD 2002. Madison, WI, USA (June 2002)
38. Jhawar, R., Piuri, V.: Fault tolerance management in IaaS clouds. In: Proc. of ESTEL 2012. Rome, Italy (October 2012)
39. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of CSE 2012. Paphos, Cyprus (December 2012)
40. Kärger, P., Olmedilla, D., Balke, W.T.: Exploiting preferences for minimal credential disclosure in policy-driven trust negotiations. In: Proc. of SDM 2008. Auckland, New Zealand (August 2008)
41. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proc. of SIGMOD 2006. Chicago, IL, USA (June 2006)
42. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: Proc. of VLDB 2003. Berlin, Germany (September 2003)
43. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM TOS 2(2), 107–138 (May 2006)
44. Samarati, P., De Capitani di Vimercati, S.: Data protection in outsourcing scenarios: Issues and directions. In: Proc. of ASIACCS 2010. Beijing, China (April 2010)
45. Wang, H., Lakshmanan, L.: Efficient secure query evaluation over encrypted XML databases. In: Proc. of VLDB 2006. Seoul, Korea (September 2006)
46. Wang, H., Yin, J., Perng, C., Yu, P.: Dual encryption for query integrity assurance. In: Proc. of CIKM 2008. Napa Valley, CA, USA (October 2008)
47. Williams, P., Sion, R.: Single round access privacy on outsourced storage. In: Proc. of CCS 2012. Raleigh, NC, USA (October 2012)
48. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In: Proc. of CCS 2008. Alexandria, VA, USA (October 2008)
49. Wong, C., Gouda, M., Lam, S.: Secure group communications using key graphs. IEEE/ACM TON 8(1), 16–30 (February 2000)
50. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proc. of VLDB 2007. Vienna, Austria (September 2007)
51. Xie, M., Wang, H., Yin, J., Meng, X.: Providing freshness guarantees for outsourced databases. In: Proc. of EDBT 2008. Nantes, France (March 2008)
52. Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: Proc. of SIGMOD 2009. Providence, RI, USA (June-July 2009)
53. Yao, D., Frikken, K., Atallah, M., Tamassia, R.: Private information: To reveal or not to reveal. ACM TISSEC 12(1), 1–27 (October 2008)