Chapter 1

Database security and privacy

Sabrina De Capitani di Vimercati[1], Sara Foresti[1], Sushil Jajodia[2], Pierangela Samarati[1]

[1] DI - Università degli Studi di Milano, Italy
[2] CSIS - George Mason University, USA

1.1 Introduction

In the last few years, the wide availability of computational and storage resources at low prices has substantially changed the way in which data are managed, stored, and disseminated. As testified by the growing success of data outsourcing, cloud computing, and services for sharing personal information (e.g., Flickr, YouTube, Facebook), both individuals and companies are more and more resorting to external third parties for the management, storage, and (possibly selective) dissemination of their data. This practice has several advantages with respect to the in-house management of the data. First, the data owner needs neither to buy expensive hardware and software licenses nor to hire skilled personnel for managing her data, thus having economic advantages. Second, the external server guarantees high data availability and highly effective disaster protection. Third, even private individuals can take advantage of the avant-garde hardware and software resources made available by providers to store, elaborate, and widely disseminate large data collections (e.g., multimedia files). The main problem of this outsourcing trend is that the data owner loses control over her data, thus increasing security and privacy risks. Indeed, the data stored at an external server may include sensitive information that the external server (or users accessing them) are not allowed to read. The specific security and privacy issues that need to be considered vary depending on the main goal for which the data owner provides her data to a third party. In particular, we identify two scenarios: a *data outsourcing* scenario

where the data owner delegates the management and storage of a data collection, possibly including sensitive information that can be selectively accessed by authorized users, to a *honest-but-curious* external server; and a *data publishing* scenario where the data owner delegates the storage of a data collection to an external server for its public dissemination. A honest-but-curious server is typically trusted to properly manage the data and make them available when needed but it may not be trusted by the data owner to read data content. Both these scenarios are characterized by the interactions among four parties: *data owner*, an organization (or an individual) who outsources her data to an external server; *user*, an individual who can access the data; *client*, the user's front-end in charge of translating access requests formulated by the user in equivalent requests operating on the outsourced data; *server*, the external third party that stores and manages the data.

The goal of this chapter is to provide an overview of the main data security and privacy issues that characterize the two scenarios mentioned above along with possible approaches for their solution (Sections 1.2 and 1.3). For each problem considered, we also briefly mention some open issues that still need further consideration and analysis. Clearly, since the data outsourcing and data publishing scenarios have some similarities, we also describe the main issues that are common to the two scenarios (Section 1.4). In the discussion, for simplicity, but without loss of generality, we assume that the outsourced data are stored in a single relation r, defined over relational schema $R(a_1, \ldots, a_n)$, which includes all sensitive information that needs to be protected. The problems as well as the approaches for their solution that we will describe in the following can however be applied to any data model (e.g., XML data or arbitrary set of resources).

1.2 Security issues in the data outsourcing scenario

The security issues specifically characterizing the data outsourcing scenario are related to three main problems that will be discussed in the following: *i)* data confidentiality, *ii)* efficient evaluation of users' queries at the server-side, and *iii)* access control enforcement.

1.2.1 Data confidentiality

Collected data often include sensitive information whose protection is mandatory, as also testified by recent regulations forcing organizations to provide privacy guarantees when storing, processing, or sharing their data with others (e.g., [9, 59, 60]). Since in the data outsourcing scenario these data are not under the direct control of their owners, individuals as well as companies require the protection of their sensitive information not only against external users breaking into the system, but also against malicious insiders, including the storing server. In many cases, the storing server is assumed to be *honest-but-curious*, that is, it is relied upon for ensuring availability of data but it is not allowed to read their content. Ensuring effective and practical data protection in this scenario is complex and requires the design of approaches allowing data owners to specify privacy requirements on data, as well as techniques for enforcing them.

Solutions. The first approach proposed to provide confidentiality of outsourced data consists in wrapping a protective layer of encryption around sensitive data to counteract both outside attacks and the curiosity of the server itself (e.g., [10, 36, 39, 63]).

Encryption represents an effective approach to guarantee proper confidentiality protection. However, since the server is not authorized to decrypt outsourced data, it cannot eval-

PATIENTS

SSN	Name	DoB	ZIP	Race	Disease	Doctor
123456789	Alice	1980/02/10	22010	asian	flu	I. Smith
234567891	Bob	1980/02/15	22018	asian	gastritis	J. Taylor
345678912	Carol	1980/04/15	22043	asian	flu	K. Doe
456789123	David	1980/06/18	22040	white	hypertension	L. Green
567891234	Erik	1980/06/18	22043	white	asthma	K. Doe
678912345	Frank	1980/10/07	22015	white	HIV	L. Green
789123456	Gary	1980/10/15	22010	white	HIV	L. Green
891234567	Hellen	1980/10/28	22018	black	flu	I. Smith

(a)

c_1={SSN}
c_2={Name,Disease}
c_3={Name,Doctor}
c_4={DoB,ZIP,Race}
c_5={Disease,Doctor}

(b)

FIGURE 1.1: An example of relation (a) and of a set of constraints over it (b)

uate users' queries. To minimize the use of encryption and make access to outsourced data more efficient, recent proposals combine *fragmentation* and *encryption* techniques [1, 14, 15]. These approaches are based on the observation that often data are not sensitive per se but what is sensitive is their association with other data. It is therefore sufficient to protect sensitive associations to preserve data confidentiality. The privacy requirements characterizing the outsourced data collection are modeled through *confidentiality constraints* [1]. A confidentiality constraint c over a relational schema $R(a_1,\ldots,a_n)$ is a subset of attributes in R, meaning that for each tuple in r, the (joint) visibility of the values of the attributes in c is considered sensitive and must be protected. As an example, Figure 1.1(b) illustrates a set of five confidentiality constraints for relation PATIENTS in Figure 1.1(a), stating that: the list of Social Security Numbers is considered sensitive (c_1); the associations of patients' names with the diseases they suffer from or their caring doctor are considered sensitive (c_2 and c_3); the association among date of birth, ZIP, and race is considered sensitive (c_4); and the association between the disease of a patient and her caring doctor is considered sensitive (c_5).

Given a relation r and a set $\mathcal{C}$ of confidentiality constraints over it, the goal is to outsource the content of r in such a way to obfuscate sensitive associations. The idea is to encrypt the sensitive attributes by making them non-intelligible to non-authorized users, and to break sensitive associations by partitioning the attributes in R in different subsets (fragments) that cannot be joined by non-authorized users. 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.

The approaches proposed in the literature combining fragmentation and encryption to protect data confidentiality differ in how the original relational schema R is fragmented and in how and whether encryption is used. In particular, existing approaches can be classified as follows.

- *Non-communicating pair of servers* [1]. R is partitioned into two fragments stored at two non-communicating servers. Encryption is used to protect attributes that cannot be stored at any of the two servers without violating constraints. For instance, consider relation PATIENTS in Figure 1.1(a) and the confidentiality constraints in Figure 1.1(b). A correct fragmentation is represented by: F_1={tid, Name, DoB, ZIP, SSN^k, Doctor^k}, F_2={tid, Race, Disease, SSN^k, Doctor^k}, where F_1 is stored at the first server, F_2 is stored at the second server. Attribute tid is the tuple identifier, introduced in both fragments to guarantee the lossless join between F_1 and F_2. Attributes SSN^k and Doctor^k contain the encrypted version of attributes SSN and Doctor, respectively.
- *Multiple fragments* [15]. R is partitioned into an arbitrary number of disjoint fragments (i.e., with no common attribute), possibly stored at a same server. Each fragment

F_1

salt	Name	DoB	ZIP	enc
s_{11}	Alice	1980/02/10	22010	bNh67!
s_{12}	Bob	1980/02/15	22018	tr354'
s_{13}	Carol	1980/04/15	22043	7feW[0
s_{14}	David	1980/06/18	22040	(uhs3C
s_{15}	Erik	1980/06/18	22043	@l3WcX
s_{16}	Frank	1980/10/07	22015	2Xdc6?
s_{17}	Gary	1980/10/15	22010	)2okED
s_{18}	Hellen	1980/10/28	22018	=ieDc2

F_2

salt	Race	Disease	enc
s_{21}	asian	flu	wEqp8
s_{22}	asian	gastritis	Ap9yt4
s_{23}	asian	flu	vl:3rP
s_{24}	white	hypertension	tgz08/
s_{25}	white	asthma	erLK;-
s_{26}	white	HIV	?(iRo4
s_{27}	white	HIV	+)ie5X
s_{28}	black	flu	Ghi3*;

F_3

salt	Doctor	enc
s_{31}	I. Smith	5tihD]
s_{32}	J. Taylor	rtF56.
s_{33}	K. Doe	se-D4C
s_{34}	L. Green	eF6hjN
s_{35}	K. Doe	3Ghv8V
s_{36}	L. Green	ee%pl;
s_{37}	L. Green	Kjh4br
s_{38}	I. Smith	+ihE67

FIGURE 1.2: An example of correct fragmentation in the multiple fragments scenario

includes a subset of the original attributes in the clear, and all the other attributes in encrypted form. For instance, consider relation PATIENTS in Figure 1.1(a) and the confidentiality constraints in Figure 1.1(b). A correct fragmentation is represented by: F_1={salt, Name, DoB, ZIP, enc}; F_2={salt, Race, Disease, enc}; and F_3={salt, Doctor, enc} in Figure 1.2. Here, salt is a randomly chosen value different for each tuple in each fragment, and enc is the result of the encryption of the attributes in PATIENTS not appearing in the clear in the fragment, concatenated with the salt.

- *Departing from encryption* [14]. R is partitioned into two fragments, one stored at the data owner side and one stored at the server side, which can be joined by authorized users only. For instance, consider relation PATIENTS in Figure 1.1(a) and the confidentiality constraints in Figure 1.1(b). A correct fragmentation is represented by: F_o={tid, SSN, Name, Race, Disease}, stored at the data owner side; and F_s={tid, DoB, ZIP, Doctor}, stored at the storing server side. Note that F_o can include in the clear all the attributes composing a constraint since it is stored at a trusted party.

Open issues. Different open issues still remain to be addressed to effectively and efficiently provide confidentiality of outsourced data. For instance, the fragmentation process should take into account dependencies among attributes in the original relation. In fact, dependencies could be exploited by observers to reconstruct the association among attributes appearing in different fragments. As an example, the specialty of a patient's doctor may reveal the disease the patient suffers from (e.g., an oncologist takes care of people suffering from cancer). Furthermore, all the proposals in the literature only protect the confidentiality of static datasets. In real world scenarios, however, outsourced data collections are subject to frequent changes that should be carefully managed to prevent information leakage. As an example, the insertion of a tuple into relation PATIENTS in Figure 1.1(a) translates into the insertion of a tuple into each of the fragments in Figure 1.2. An observer can therefore easily reconstruct the sensitive associations among the attribute values for the new tuple.

1.2.2 Efficient query evaluation

In the data outsourcing scenario, the storing server does not have complete visibility of the outsourced data, as they are fragmented and/or encrypted to preserve confidentiality. Therefore, it cannot evaluate users' queries. Also, neither the data owner should be involved in the query evaluation process (this would nullify the advantages of data outsourcing), nor the client should download the complete outsourced data collection to locally evaluate queries. It is therefore necessary to define techniques that permit to partially delegate to the external server the query evaluation process, while not opening the door to inferences.

Solutions. In the last few years, several techniques have been proposed to support the server-side evaluation of a wide set of selection conditions and SQL clauses when the

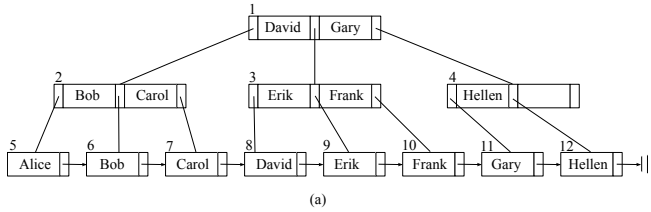
PATIENTS^k

id	enc	$I_{\texttt{DoB}}$	$I_{\texttt{ZIP}}$	$I_{\texttt{Race}}$
1	zKZlJxV	α	δ	η
2	AJvaAy1	α	ϵ	η
3	AwLBAa1	α	ζ	η
4	mHF/hd8	β	δ	θ
5	HTGhoAq	β	ζ	θ
6	u292mdo	γ	ζ	θ
7	ytOJ;8r	γ	δ	θ
8	eWo09uH	γ	ϵ	ι

FIGURE 1.3: An example of encrypted relation with indexes

outsourced relation is completely encrypted. These solutions complement the encrypted relation with additional metadata, called *indexes*, on which queries are evaluated at the server side. Relation r, defined over schema $R(a_1, \ldots, a_n)$, is then mapped to an encrypted relation r^k over schema $R^k(\underline{\texttt{tid}}, \texttt{enc}, I_{i_1}, \ldots, I_{i_j})$, with `tid` a numerical attribute added to the encrypted relation and acting as a primary key for R^k; `enc` the encrypted tuple; $I_{i_l}, l = 1, \ldots, j$, the index associated with the i_l-th attribute a_{i_l} in R. For instance, Figure 1.3 illustrates the encrypted version of relation PATIENTS in Figure 1.1(a), assuming the presence of indexes for `DoB` ($I_{\texttt{Dob}}$), `ZIP` ($I_{\texttt{ZIP}}$), and `Race` ($I_{\texttt{Race}}$). For readability, index values are represented with Greek letters and we report the tuples in the plaintext and encrypted relations in the same order. Note, however, that the order in which tuples are stored in the encrypted relation is independent from the order in which they appear in the plaintext relation. Different indexing techniques support different types of queries. In the following, we illustrate some techniques, partitioning them according to the queries that each of them can manage.

- *Equality conditions* (e.g., [19, 39]). The first approach proposed to support equality conditions is represented by *encryption-based indexes* [19]. Given a tuple t, the value of the index for attribute a is computed as $E_k(t[a])$, where E_k is a symmetric encryption function and k the encryption key. As a consequence, any condition of the form $a{=}v$ is translated as $I_a{=}E_k(v)$. For instance, consider index $I_{\texttt{Race}}$ in Figure 1.3 obtained by adopting this method. Then, condition `Race` = 'asian' on relation PATIENTS is translated as $I_{\texttt{Race}} = E_k(\text{asian})$, that is, $I_{\texttt{Race}}{=}$'η' on relation PATIENTS^k. An alternative solution is represented by *bucket-based indexes* [39]. The domain of attribute a is partitioned into non-overlapping subsets of contiguous values and each partition is associated with a label. Given a tuple t in the outsourced relation r, the value of the index associated with attribute a is the label of the unique partition containing value $t[a]$. As a consequence, equality condition $a{=}v$ is translated as $I_a{=}l$, where l is the label of the partition including v. For instance, index $I_{\texttt{DoB}}$ in Figure 1.3 has been obtained by partitioning the domain [1980/01/01, 1980/12/31] of attribute `DoB` in intervals of 4 months, and assigning, in the order, labels α, β, and γ to the three partitions. Condition `DoB` = 1980/04/15 on relation PATIENTS is translated as $I_{\texttt{DoB}} =$ 'β' on PATIENTS^k, since β is the label for the partition [1980/04/01, 1980/07/31]. A third technique efficiently supporting equality conditions is represented by *hash-based indexes* [19]. Given a tuple t in r, the value of the index associated with attribute a is computed as $h(t[a])$, where h is a deterministic hash function that generates collisions. Therefore, condition $a{=}v$ is translated as $I_a{=}h(v)$. For instance, consider index $I_{\texttt{ZIP}}$ in Figure 1.3 obtained by adopting this method. Then, condition `ZIP` = 22010 on relation PATIENTS is translated as $I_{\texttt{ZIP}} = h(22010)$, that is, $I_{\texttt{ZIP}}{=}$'δ' on relation PATIENTS^k. Both bucket-based and hash-based indexes map different plaintext



(a)

id_node	node
1	2, David, 3, Gary, 4
2	5, Bob, 6, Carol, 7
3	8, Erik, 9, Frank, 10
4	11, Hellen, 11
5	Alice, 6, t_1
6	Bob, 7, t_2
7	Carol, 8, t_3
8	David, 9, t_4
9	Erik, 10, t_5
10	Frank, 11, t_6
11	Gary, 12, t_7
12	Hellen, NIL, t_8

(b)

id	content
1	8/*5sym,p
2	mw39wio[
3	gtem945/*c
4	21!p8dq59
5	8dq59wq*d'
6	ue63/)w
7	=wco21!ps
8	oieb5(p8*
9	gte3/)8*
10	rfoi7/(
11	=o54'?c
12	Fer3!-r

(c)

FIGURE 1.4: An example of B+-tree index (a), its relational representation (b), and the corresponding encrypted relation (c)

values to the same index value. Therefore, the result computed by the server may include *spurious tuples* that the client will filter out by executing a query that evaluates the original condition on the tuples received from the server (after their decryption). For instance, only two of the three tuples in the encrypted relation with value 'δ' for attribute $I_{\texttt{ZIP}}$ satisfy condition `ZIP` = 22010.

- *Range conditions* (e.g., [2, 19, 64]). To overcome the limitations of indexing techniques that support only equality conditions, in [19] the authors present a B+-tree index that allows the evaluation of both equality and range queries at the server side. The B+-tree index is built by the data owner over the original plaintext values of an attribute. It is then represented as a relational table, encrypted, and stored at the server. This relation is iteratively queried by the client for retrieving the tuples satisfying the query condition. For instance, Figure 1.4 illustrates the B+-tree index built for attribute `Name` of relation PATIENTS in Figure 1.1(a), the relation representing it, and the encrypted relation stored at the external server. Condition `Name` LIKE '[E–Z]%' on relation PATIENTS (retrieving the names following 'E' in lexicographic order) is evaluated by executing a set of queries for traversing the B+-tree along the path of nodes 1, 3, and 9, and to follow the chain of leaves starting at node 9. For each visited leaf, the client retrieves the tuple associated with it (i.e., tuple t_5 for leaf 9, tuple t_6 for leaf 10, tuple t_7 for leaf 11, and tuple t_8 for leaf 12). To avoid to store additional relations on the server, *order preserving encryption indexes* have recently been proposed [2, 64]. These techniques are based on order preserving encryption schemas, which take as input a target distribution of index values and apply an order preserving transformation in a way that the transformed values (i.e., the index values) follow the target distribution.

- *Aggregate operators* (e.g., [33, 38]). Privacy homomorphic encryption [61] allows the execution of basic arithmetic operations (i.e., $+,-,\times$) directly over encrypted data. Their adoption in the definition of indexes allows the server to evaluate aggregate functions and to execute equality and range queries. The main drawback of these approaches is their computational complexity, which makes them not suitable for many real-world applications.

The main challenge that must be addressed in the definition of indexing techniques is balancing precision and privacy: more precise indexes provide more efficient query execution, at the price of a greater exposure to possible privacy violations. As an example,

encryption-based indexes permit to completely delegate the evaluation of equality conditions to the server. However, index values have exactly the same frequency distribution as plaintext values, thus opening the door to frequency-based attacks. For instance, it is easy to see that value ι for index $I_{\texttt{Race}}$ represents value *black* for attribute `Race`, since *black* (ι, respectively) is the only value with one occurrence for attribute `Race` (index $I_{\texttt{Race}}$, respectively). Analogously, a higher number of indexes on the same relation improves query evaluation efficiency, while causing a higher risk of inference and linking attacks. As shown in [10], even a limited number of indexes can greatly facilitate the task for an adversary who wants to violate the confidentiality provided by encryption.

The evaluation of queries over outsourced data requires the definition of proper techniques defining how queries on the original table are translated into queries on the encrypted data and indexes over them, or on fragmented data. For instance, consider relation PATIENTS in Figure 1.1(a) and its fragmentation in Figure 1.2. Query " SELECT `Name` FROM PATIENTS WHERE `Disease`='flu' " is translated into the following query operating on fragment F_2, where attribute `Disease` is represented in clear from: " SELECT `enc` FROM F_2 WHERE `Disease`='flu' ". When the client receives the query result, it decrypts attribute `enc` and applies a projection on patients' names before showing the result to the requesting user. Query plans are designed to minimize the client's overhead in query evaluation. In fact, most query evaluations will require the client's intervention since the server cannot decrypt encrypted attributes. The techniques designed to define efficient query plans depend on the fragmentation approach adopted (e.g., [13, 15, 31]).

Open issues. Besides the definition of alternative indexing techniques for encrypted data and for efficiently evaluating queries on fragmented data, it still remains to study the possibility of combining fragmentation and indexing approaches. In fact, fragmentation does not permit to delegate the evaluation of conditions involving attributes that do not appear plaintext in a fragment. The association of indexes to fragments could nicely fill this gap, but should be carefully designed to prevent information leakage caused by the plaintext representation in a fragment of an attribute indexed in another.

1.2.3 Access control enforcement

In most real-world systems, access to the data is selective, that is, different users enjoy different views over the data. When the data are managed by an external third party, the problem of how to enforce the access control policy defined by the owner becomes crucial. In fact, the data owner cannot enforce the access control policy since this would imply that the data owner has to mediate every access request, thus losing advantages of outsourcing data. Also, access control enforcement cannot be delegated to the external server as in traditional systems (e.g., [6]) for privacy reasons. Indeed, the access control policy might be sensitive, and the server might collude with malicious users to gain access to sensitive data. Therefore, it is necessary to define techniques that permit the data themselves to enforce access control policies.

Solutions. The solutions proposed to enforce access control restrictions on outsourced data without the data owner's intervention are based on integrating access control and encryption.

The enforcement of read privileges has been addressed first and has been efficiently solved by mapping selective visibility restrictions in the encryption of the data [23]. The authorization policy is translated into an equivalent *encryption policy*, regulating which data are encrypted with which key and which keys are released to which users. This translation process is performed considering the following two desiderata: *i)* at most one key is released to each user, and *ii)* each tuple is encrypted at most once. To this purpose, the approach in [23] adopts a key derivation method, which permits to compute an encryption key k_i

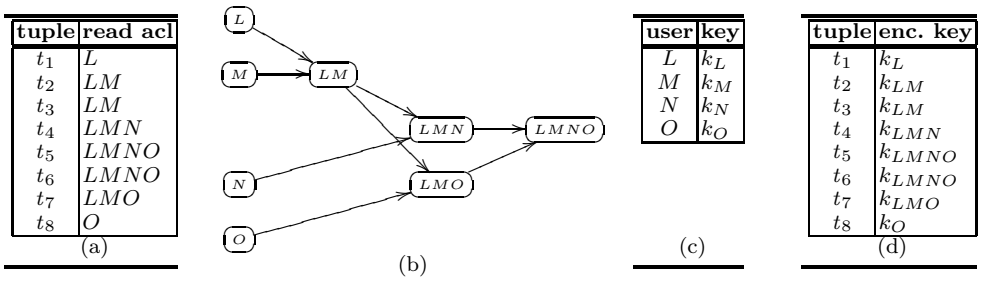


FIGURE 1.5: An example of access control policy (a), the corresponding key derivation graph (b), keys assigned to users (c), and keys used to encrypt resources (d)

starting from the knowledge of another key k_j and a piece of publicly available information [4]. Key derivation methods are based on the definition of a key derivation hierarchy that specifies which keys can be derived from other keys in the system. A key derivation hierarchy that correctly enforces the access control policy must permit each user to derive from her key all and only the keys used to encrypt the tuples she can access. To this aim, the hierarchy has a node (which represents a key) for each user in the system and a node for each access control list (acl), that is, for each set of users who can access a tuple. The edges in the hierarchy (which correspond to key derivation operations) guarantee the existence of a path connecting each node representing a user to each node representing a set to which the user belongs [23]. Each user knows the key of the node representing herself in the hierarchy and each resource is encrypted with the key representing its access control list. For instance, consider a system with four users $\{L,M,N,O\}$ and the tuples composing relation PATIENTS in Figure 1.1(a). Figure 1.5(b) reports the key derivation graph enforcing the access control policy in Figure 1.5(a), and Figures 1.5(c,d) summarize the keys communicated to the users and the keys used to encrypt the tuples. For simplicity and readability, in the key derivation graph nodes are labeled with the set of users they represent. It is easy to see that each user knows or can derive all and only the keys of the nodes including herself. As a consequence, she can decrypt all and only the tuples in relation PATIENTS that she is authorized to access (i.e., such that the user belongs to the acl of the tuple).

The solution in [25] complements the technique in [23] by associating a write token with each tuple. The write token is encrypted with a key that only users who can modify the tuple and the external server can derive. The server accepts a write operation only if the requesting user proves to be able to correctly decrypt the write token associated with the modified tuple. For instance, Figure 1.6(a) illustrates the access control policy in Figure 1.5(a) for relation PATIENTS, extended with write privileges on the tuples. Figure 1.6(b) illustrates the key derivation graph, which extends the graph in Figure 1.5(b) introducing the external server $\mathcal{S}$. Figures 1.6(c,d) summarize the keys communicated to the users and the keys used to encrypt the tuples and the corresponding write tokens.

The enforcement of access control restrictions through encryption raises many other issues that have been recently addressed, as summarized in the following.

- Since the key used to encrypt each tuple depends on the set of users who can access it, updates to the access control policy require data re-encryption, which represents an expensive operation for the data owner. For instance, consider the example in Figure 1.5 and assume that user O is revoked access to tuple t_5. To enforce such a revocation, the data owner should download tuple t_5, decrypt it using key k_{LMNO}, re-encrypt the tuple with key k_{LMN}, and send it back to the server. To prevent re-encryption, while enforcing updates to the access control policy, the proposal in [23]

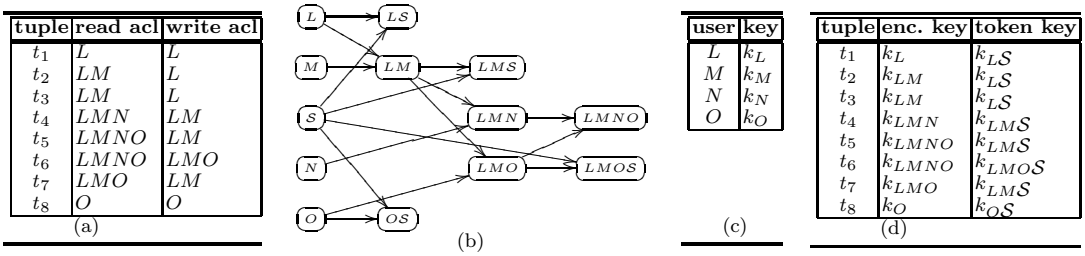


FIGURE 1.6: An example of read and write access control policy (a), the corresponding key derivation graph (b), keys assigned to users (c), and keys used to encrypt resources (d)

adopts two layers of encryption: one managed by the data owner and enforcing the initial policy (BEL); and one managed by the external server and enforcing policy updates (SEL). A user can then access a tuple only if she knows (or can derive) the key used for encrypting the resources at both levels.

- The adoption of key derivation for access control enforcement may reveal to the server the authorization policy. Whenever the policy is considered sensitive, the structure of the key derivation hierarchy should not be revealed to the external server, as proposed in [21].

- Many real-world scenarios are characterized by parties acting both as data producers (i.e., data owners) and as data consumers (i.e., final users). The enforcement of access control should then take into consideration the fact that there are multiple data owners, each regulating access to her data [22].

- The combined use of selective encryption (for access control enforcement) and indexing techniques (for efficient query evaluation) may open the door to inference on attribute values of tuples that users are not authorized to read [24]. In fact, users have also visibility on indexes of tuples they are not allowed to access. Since index values depend on the attribute value they represent, such a visibility could permit users to infer attribute values for tuples they cannot access. For instance, consider plaintext relation PATIENTS in Figure 1.1(a), its encrypted version in Figure 1.3, and the access control policy in Figure 1.5(a). User M, who is authorized to read tuple t_2, can easily infer that value η for index $I_{\texttt{Race}}$ represents plaintext value 'asian'. As a consequence, M can conclude that t_1[Race]='asian', thus breaching the confidentiality of tuple t_1 that she is not authorized to access. To limit this risk, the indexing function should be designed to take access control restrictions into consideration [24].

An alternative solution to the combined use of selective encryption and key derivation for access control enforcement has been introduced in [71]. This approach aims at providing systems scalability by adopting *attribute-based encryption*. In this scenario, the traditional access control policy is substituted by the definition of a set of attributes associated with users and tuples regulating which user can access which tuple. More precisely, each tuple is associated with a set of attributes that describe the context in which the tuple should be accessed, and each user is associated with a logical expression over the attributes that represents the properties of the tuples she is authorized to access. To enforce the access control policy, attribute-based encryption techniques define a different public key component for each attribute in the system. Each tuple is then encrypted using a key that reflects all the public key components of the attributes associated with it, while each user knows the secret key that permits to decrypt all the tuples she is authorized to access.

Open issues. Most of the problems studied for the enforcement of read privileges need to be analyzed also for the solutions enforcing write privileges. For instance, it is necessary to define an efficient approach for managing policy updates without the intervention of the data owner, and a technique for protecting the confidentiality of the policy when read and write operations are restricted to arbitrary subsets of users.

1.3 Security issues in the data publishing scenario

In the data publishing scenario, information about entities, called *respondents*, are publicly or semipublicly released. While in the past released information was mostly in tabular and statistical form (*macrodata*), many situations require today the release of specific data (*microdata*). Microdata provide the convenience of allowing the final recipient to perform on them analysis as needed, at the price of a higher risk of exposure of respondents' private information. In this section, we describe how public data release may cause the unintended disclosure of respondents' identities and/or sensitive attributes, and the solutions proposed to counteract this risk.

1.3.1 Preserving respondents privacy

To protect respondents' identities, data owners remove or encrypt explicit identifiers (e.g., SSN, name, phone numbers) before publishing their datasets. However, data *de-identification* provides no guarantee of anonymity since released information often contains other data (e.g., race, birth date, sex, and ZIP code) that can be linked to publicly available information to re-identify (or restrict the uncertainty about the identity of) data respondents, thus leaking information that was not intended for disclosure. The large amount of information easily accessible today, together with the increased computational power available to data recipients, make such linking attacks easier [35]. Furthermore, the disclosure of an individual's identity (*identity disclosure*) often implies also the leakage of her sensitive information (*attribute disclosure*). For instance, in 2006 Netflix, an online DVD delivery service, started a competition whose goal was the improvement of its movie recommendation system based on users' previous ratings. To this purpose, Netflix released 100 million records about movie ratings of 500,000 of its subscribers. The released records were anonymized removing the personal identifying information of the subscribers, which was substituted with an anonymous customer id. However, by linking the movie recommendations available on the Internet Movie Database (IMDb) with the anonymized Netflix dataset, it was possible to re-identify individuals, thus revealing potentially sensitive information (e.g., apparent political preferences) [55].

Protecting respondents' identities and sensitive attributes in the today's global interconnected society is a complex task that requires the design of effective techniques that permit data owners to provide privacy guarantees, even without knowing the external sources of information that a possible observer could exploit to re-identify data respondents. Note that the public release of a dataset should also provide utility for data recipients (i.e., the released dataset should include as much information as possible to permit final recipients to obtain representative results from their analysis). Clearly, data utility and privacy are two conflicting requirements that need to be balanced in data release.

Solutions. The solutions proposed to protect respondents' privacy, while providing data recipients with useful data, can be classified in the following two categories.

PATIENTS

SSN	Name	DoB	ZIP	Race	Disease	Doctor
		1980/02	2201*	asian	flu	I. Smith
		1980/02	2201*	asian	gastritis	J. Taylor
		1980/06	2204*	white	hypertension	L. Green
		1980/06	2204*	white	asthma	K. Doe
		1980/10	2201*	white	HIV	L. Green
		1980/10	2201*	white	HIV	L. Green

FIGURE 1.7: An example of 2-anonymous table

- *Approaches based on k-anonymity* (e.g., [46, 49, 62]) guarantee that the dataset publicly released satisfies properties that provide protection against identity and attribute disclosure (e.g., every combination of values of quasi-identifiers can be indistinctly matched to at least k respondents).

- *Approaches based on differential privacy* (e.g., [30, 41]) guarantee that the released dataset is protected against certain kinds of inference defined before data release (e.g., an observer cannot infer with non-negligible probability whether a subject is represented in the released dataset).

The first approach proposed in the literature to protect respondents' privacy in microdata release is represented by *k-anonymity* [62]. k-anonymity captures the well-known requirement, traditionally applied by statistical agencies, stating that any released data should be indistinguishably related to no less than a certain number of respondents. Since linking attacks are assumed to exploit only released attributes that are also externally available (called *quasi-identifiers*), in [62] this general requirement has been translated into the following k-anonymity requirement: *Each release of data must be such that every combination of values of quasi-identifiers can be indistinctly matched to at least k respondents*. A microdata table then satisfies the k-anonymity requirement if each tuple in the table cannot be related to less than k respondents in the population and vice versa (i.e., each respondent in the population cannot be related to less than k tuples in the released table). The k-anonymity requirement can be checked only if the data owner knows any possible external source of information that may be exploited by a malicious recipient for respondents re-identification. This assumption is, however, limiting and highly impractical in most scenarios. k-anonymity then takes a safe approach by requiring that each combination of values of the quasi-identifier attributes appears with at least k occurrences in the released table, which is a sufficient (although not necessary) condition to satisfy the k-anonymity requirement. Traditional approaches for guaranteeing k-anonymity transform the values of the attributes composing the quasi-identifier, while leaving sensitive attribute values unchanged. To guarantee truthfulness of released data, k-anonymity relies on *generalization* and *suppression* microdata protection techniques [17]. Generalization consists in substituting the original values with more general values (e.g., the date of birth can be generalized by removing the day, or the day and the month, of birth). Suppression consists in removing data from the microdata table. The combined use of these techniques guarantees the release of a less precise and less complete, but truthful, data while providing protection of respondents' identities. As an example, the table in Figure 1.7 has been obtained from the table in Figure 1.1(a) by: *i)* removing explicit identifiers (i.e., `SSN` and `Name`); *ii)* generalizing attribute `DoB` removing the day of birth; *iii)* generalizing attribute `ZIP` removing the last digit; and *iv)* suppressing the third and eight tuples in the original table. The resulting table

is 2-anonymous since each combination of values for attributes DoB, Race, and ZIP appears (at least) twice in the relation. To limit the information loss caused by generalization and suppression (i.e., to improve utility of released data), many k-anonymity algorithms have been proposed (e.g., [5, 16, 42, 43, 62]). All these approaches are aimed at minimizing the loss of information caused by generalization and suppression, while providing the privacy guarantees required by respondents. Although k-anonymity represents an effective solution for protecting respondents' identities, it has not been designed to protect the released microdata table against attribute disclosure. Given a k-anonymous table it may then be possible to infer (or reduce the uncertainty about) the value of the sensitive attribute associated with a specific respondent. This happens, for example, when all the tuples with the same value for the quasi-identifier are associated with the same value for the sensitive attribute. For instance, consider the 2-anonymous table in Figure 1.7 and assume that *Susan* knows that her friend *Frank* is a male, born in October 1980, and living in 22015 area. *Susan* can easily infer that *Frank*'s tuple is either the fifth or the sixth tuple in the published table. As a consequence, *Susan* can infer that her friend suffers from HIV. ℓ-diversity [49] and t-closeness [46] has been proposed to protect released microdata tables against attribute disclosure.

Approaches based on the definition of differential privacy [30] have been recently proposed to guarantee that the released microdata table protects respondents against inference that exploits both published data and external adversarial knowledge. One of the first definitions of privacy in the data publishing scenario states that: *Anything that can be learned about a respondent from the statistical database should be learnable without access to the database* [18]. Although this definition has been thought for statistical databases, it is also well suited for the microdata publishing scenario. However, the main problem of this definition of privacy is that only an empty dataset can guarantee absolute disclosure prevention [30]. Differential privacy can be considered as the first attempt of achieving privacy as defined in [18]. It is based on the observation that the release of a dataset may violate the privacy of any individual, independently of whether she is represented in the dataset. For instance, suppose that the released dataset permits to compute the average annual benefits of people living in 22010 area for each ethnic group, and suppose that this information is not publicly available. Assume also that *Alice* knows that *Bob*'s annual benefits is 1,000$ more than the average annual benefits of Asian people living in 22010 area. Although this piece of information alone does not permit *Alice* to gain any information about *Bob*'s annual benefits, when it is combined with the released dataset, it allows Alice to infer *Bob*'s annual benefits. Differential privacy aims at preventing an observer from inferring whether a subject is represented or not in the released dataset. A data release is then considered safe if the inclusion in the dataset of tuple t_p, related to respondent p, does not change the probability that a malicious recipient can correctly identify the sensitive attribute value associated with p. Intuitively, differential privacy holds if the removal (insertion, respectively) of one tuple t_p from (into, respectively) the table does not substantially affect the result of the evaluation of a function $\mathcal{K}$ on the released table. Most of the techniques proposed to satisfy differential privacy are based on *noise addition*, which does not preserve the truthfulness of released data (e.g., [40, 44, 51, 67]) and may therefore not be suited to different data publishing scenarios. Differential privacy provides protection against inferences on the presence/absence in the published dataset of the record representing a respondent. However, this is not the only cause of privacy breaches. To provide a wider privacy guarantee, a recent line of work has put forward a definition of privacy that permits to protect released data against any kind of inference, provided it is defined (and properly modeled) by the data owner before data release [41].

Open issues. Although many efforts have been made to overcome the assumptions on which the definitions of k-anonymity and differential privacy are based, there are still open

issues that deserve further investigation. This is also testified by the fact that there is not a unique definition of respondents privacy and each definition has some drawbacks that need to be addressed (e.g., external adversarial knowledge is not adequately modeled and addressed by privacy protection techniques). Also, all the proposed solutions for protecting respondents privacy still need to be enhanced, to possibly find a good tradeoff between privacy of respondents and utility of released data for final recipients.

1.4 Security issues in emerging scenarios

In this section, we focus on the security and privacy problems that are common to both the data outsourcing scenario and the data publishing scenario. In particular, we consider the: *i)* confidentiality of users queries; *ii)* integrity of outsourced/published data; and *iii)* completeness and correctness of query results.

1.4.1 Private access

An important issue that arises when data are stored at an external server is preserving the confidentiality of users' queries, independently of whether data are kept confidential or are publicly released. As an example, assume that a user accesses an external medical database looking for the symptoms of a given disease. The server executing the query (as well as any observer) can easily infer that the user (or a person close to her) suffers from the disease in the query condition. Furthermore, queries could be exploited by the server (or by an external observer) to possibly infer sensitive information in the outsourced data collection, thus reducing the effectiveness of the techniques possibly adopted by the data owner to protect data confidentiality. Besides protecting the confidentiality of each query singularly taken, it is also important to protect *patterns* of accesses to the data (i.e., the fact that two or more queries aimed at the same target tuple). Indeed, the frequency of accesses to tuples could be exploited to breach both data and query confidentiality.

Solutions. The solutions proposed to protect the confidentiality of the queries issued by users can be classified depending if they operate on plaintext data (data publishing) or on encrypted data (data outsourcing).

The line of work first developed to protect query confidentiality is represented by classical studies on *Private Information Retrieval* (PIR) [12], which operates on plaintext datasets stored at an external server. In this model, the database is represented as a N-bit string and a user is interested in retrieving the i-th bit of the string without allowing the server to know/infer which is the bit target of her query. PIR proposals can be classified as follows: *information-theoretic* PIR, which protects query confidentiality against attackers with unlimited computing power; and *computational* PIR, which protects query confidentiality against adversaries with polynomial-time computing power. Unfortunately, information-theoretic PIR protocols have computation and communication costs linear in the size of the dataset (i.e., $\Omega(N)$) as there is no solution to the problem that is better than downloading the whole database [12]. To limit these costs, it is however possible to replicate the dataset on different servers. Intuitively, PIR approaches exploiting data replication require the user to pose a randomized query to each server storing a copy of the dataset, in such a way that neither the servers nor an outside observer can determine which is the bit to which the user is interested. The user will then reconstruct the target bit by properly combining the query results computed by the different servers [3, 12]. The main problem of these proposals is that

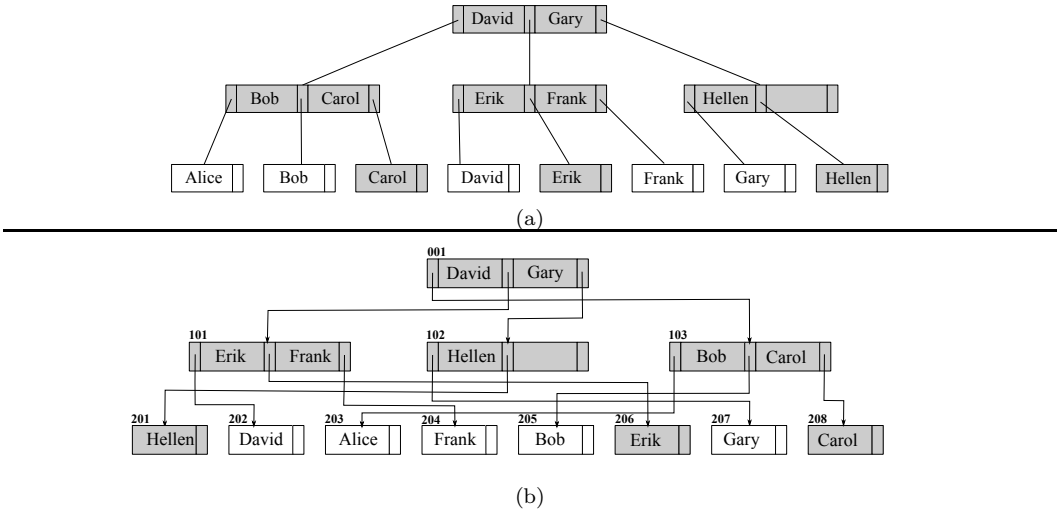


FIGURE 1.8: An example of abstract (a) and corresponding logical shuffle index (b)

they rely on the unrealistic assumption that the external servers do not communicate with each other. Computational PIR protocols exploit cryptographic properties to reduce both the communication cost and the number of copies of the data with respect to information-theoretic techniques [8, 11]. Users then adopt specific functions to encrypt their requests before submitting them to the server. These functions enjoy particular properties that allow the server to compute, without decrypting the request, an encrypted query result that only the requesting user can decrypt. Although more efficient than information-theoretic solutions, also computational PIR schemes suffer from heavy computation costs (i.e., $O(N)$ or $O(\sqrt{N})$) both for the client and for the server, which limit their applicability in real-life scenarios [32]. Recently, traditional PIR protocols have been integrated with relational databases, to the aim of protecting sensitive constant values in SQL query conditions, while providing the client with efficient query evaluation [56]. The original query formulated by the client is properly sanitized before execution, to prevent the server from inferring sensitive data. To extract the tuples of interest from the result of the sanitized query executed by the server, the client then resorts to traditional PIR protocols, operating on the sanitized query result instead of on the whole dataset (thus highly reducing computational costs).

One of the solutions operating on encrypted data, aimed at providing both data and query confidentiality, is based on the definition of a *shuffle index* on the data collection [26, 27]. A shuffle index is defined, at the abstract level, as an *unchained B+-tree* (i.e., there are no links connecting the leaves), built on one of the attributes in the outsourced relation and with actual data stored in the leaves of the tree. Figure 1.8(a) illustrates a graphical representation of the abstract data structure built on attribute `Name` of relation PATIENTS in Figure 1.1(a). At the logical level, the nodes of the tree are allocated to logical addresses that work as logical *identifiers*, which may not follow the same order of the values in the abstract nodes they represent. Figure 1.8(b) illustrates a possible representation at the logical level of the abstract data structure in Figure 1.8(a). In the figure, nodes appear ordered (left to right) according to their identifiers, which are reported on the top of each node. Pointers between nodes of the abstract data structure correspond, at the logical level, to node identifiers, which can then be easily translated at the physical level into physical addresses at the external server. For simplicity and easy reference, in our example, the first digit of the node identifier denotes the level of the node in the tree. Before sending to the

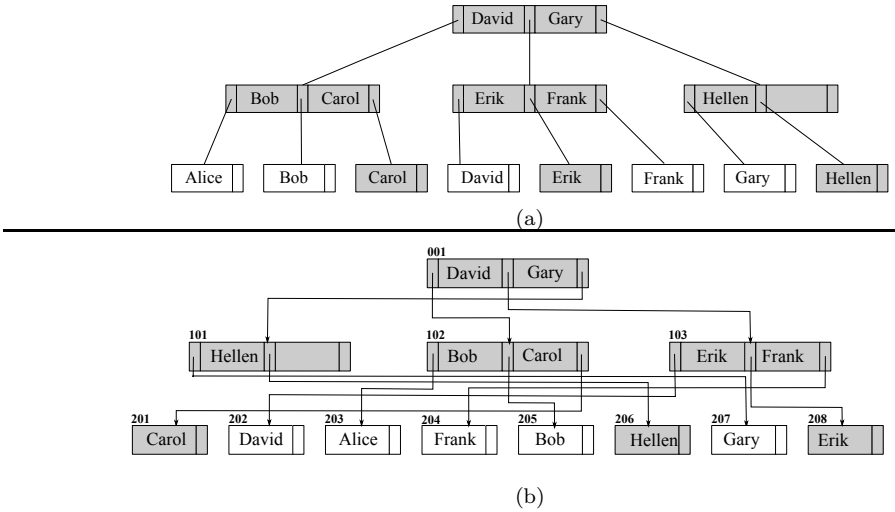


FIGURE 1.9: Abstract (a) and corresponding logical shuffle index (b) in Figure 1.8 after the execution of a search operation

server the shuffle index for storing it, the content of each node is encrypted, producing an encrypted block. Since each block is encrypted, the server does not have any information on the content of the node stored in the block or on the parent-child relationship between nodes stored in blocks. Retrieval of the leaf block containing the tuple corresponding to an index value requires an iterative process. Starting from the root of the tree and ending at a leaf, the read block is decrypted retrieving the address of the child block to be read at the next step. To protect access and pattern confidentiality, searches over the $B+$-tree are extended with the following three protection techniques [26].

- *Cover searches*: fake searches executed together with the target search. The values used as cover searches are chosen to be indistinguishable from actual target values and operate on disjoint paths of the tree. The search process retrieves, for each level in the tree, the same number of blocks (one for the target and the other for the cover searches). Cover searches introduce uncertainty over the leaf block storing the target tuple and do not allow the server to establish the parent-child relationship between blocks retrieved at contiguous levels.

- *Cache*: set of blocks/nodes along the path to the target value recently accessed, which are stored at the client-side. Cache makes searches repeated within a short time interval not recognizable as being the same search: if the nodes in the target path are already in cache, an additional cover search is executed.

- *Shuffling*: operation performed for changing the block where accessed nodes are stored, thus breaking the correspondence between nodes (contents) and blocks (addresses). Basically, the contents of blocks retrieved at each level and of the blocks at the same level stored in the local cache are mixed. Nodes are then re-encrypted and the resulting blocks rewritten accordingly on the server.

As an example, consider a search for name *Carol* over the abstract index in Figure 1.8(a) that adopts *Hellen* as cover, and assume that the local cache contains the path to *Erik* (i.e., (001,101,204)). The nodes involved in the search operation are denoted in gray in the figure.

Figure 1.9 illustrates the abstract and logical representation of the shuffle index in Figure 1.8 after the execution of the search operation, which shuffles nodes 101, 102, and 103, and nodes 201, 206, and 208.

Besides the shuffle index technique, other proposals have also been introduced to address data and query confidentiality in data outsourcing scenarios. These approaches are based on the pyramid-shaped database layout of Oblivious RAM [34] and propose an enhanced reordering technique between adjacent levels of the structure [20, 29, 47, 66]. The privacy provided by these approaches is guaranteed however by the presence of a trusted co-processor at the external server.

Open issues. Among the open issues that still need to be analyzed, there are the needs of providing efficient accesses based on the value of different attributes in the outsourced/published relation (*multiple indexes*) and of supporting possible updates to the data collection.

1.4.2 Data integrity

Traditionally, the server is assumed to be trusted to correctly store and maintain the data of the owners. There are, however, scenarios where such a trust assumption does not hold. As a consequence, the server itself or a user may modify the data collection without being authorized. Since the server directly stores and manages the dataset, it is not possible to physically prevent unauthorized data modifications, there are however techniques that permit the data owner (and any authorized user) to discover non-authorized modifications.

Solutions. Data integrity can be provided at different granularity levels: table, attribute, tuple, or cell level. The integrity verification at the table and attribute level is expensive since it can be performed by the client only downloading the whole table/column. Data integrity at the cell level suffers from a high verification overhead. For these reasons, the majority of the current proposals provide data integrity at the tuple level.

Integrity verification approaches traditionally rely on *digital signatures* (e.g., [37]), that is, the data owner first signs, with her private key, each tuple in the outsourced/published relation, and the signature is concatenated to the tuple. When a client receives a set of tuples from the external server, she can check the signature associated with each tuple to detect possible unauthorized changes. The main drawback of traditional integrity verification techniques relying on digital signature is that the verification cost at the client side grows linearly with the number of tuples in the query result.

To reduce the verification costs of large query results, in [53] the authors propose the adoption of a schema that permits to combine a set of digital signatures. In this way, the client can verify the integrity of the query result by checking one aggregated signature only: the one obtained combining the signatures of the tuples in the query result. In [53] the authors propose three different signature schemes: *condensed RSA*, based on a variation of the RSA encryption schema that allows the aggregation of signatures generated by the same signer; *BGLS* [7], based on bilinear mappings and supporting the aggregation of signatures generated by different signers; batch *DSA signature aggregation*, based on the multiplicative homomorphic property of DSA signature schema. Both condensed RSA and BGLS approaches are *mutable*, meaning that any user who knows multiple aggregated signatures can compose them, obtaining a valid aggregated signature. Although this feature can be of interest in the process of generating aggregated signatures, it also represents a weakness for the integrity of the data. In [52], the authors propose an extension of condensed RSA and BGLS techniques that makes them immutable. Such an extension is based on zero knowledge protocols and basically consists in revealing to the client a proof of the knowledge of the aggregated signature associated with the query result, instead of revealing the signature itself.

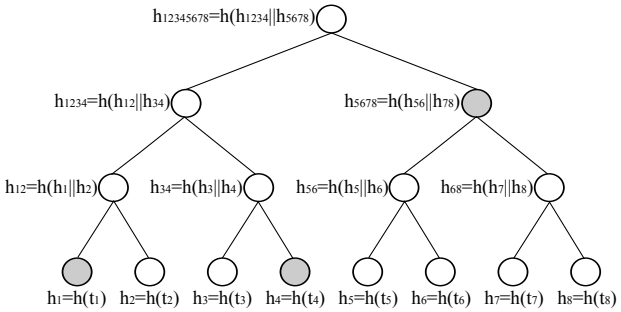


FIGURE 1.10: An example of Merkle hash tree

Open issues. Current approaches for providing integrity guarantees to externally stored data typically adopt signature techniques. These solutions are, however, computationally expensive both for the data owner and for clients. It could be useful to define alternative approaches based on less expensive techniques that permit authorized users to efficiently check data integrity.

1.4.3 Completeness and correctness of query results

Besides techniques that permit to discover unauthorized changes (data integrity in storage), it is also necessary to provide clients with methods for verifying the correctness and completeness of query results. Indeed, the external server may be lazy in computing the query result and omit tuples from the computation, or include fake tuples in the query result. The verification of query results is hard to enforce, especially in the emerging large-scale platforms used, for example, in cloud computing.

Solutions. The approaches proposed in the literature can be classified as follows.

- *Authenticated data structures* approaches (e.g., [28, 45, 50, 54, 57, 58, 70]) are based on the definition of appropriate data structures (e.g., signature chaining, Merkle hash trees, or skip lists), and provide guarantee of completeness of the result of queries operating on the attribute (set thereof) on which the data structure has been defined. These approaches also guarantee integrity of stored data since unauthorized changes to the data can be detected by the verification process of query results.
- *Probabilistic* approaches (e.g., [48, 65, 68]) are based on the insertion of sentinels in the outsourced data, which must also belong to the query result. These solutions provide a probabilistic guarantee of completeness of query results.

Most of the authenticated data structures approaches adopt *Merkle hash trees* [50]. A Merkle hash tree is a binary tree, where each leaf contains the hash of one tuple of the outsourced relation, and each internal node contains the result of the application of the same hash function on the concatenation of the children of the node itself. The root of the Merkle hash tree is signed by the data owner and communicated to authorized users. The tuples in the leaves of the tree are ordered according to the value of a given attribute a. Figure 1.10 illustrates an example of a Merkle hash tree built over relation Patients in Figure 1.1(a) for attribute `SSN`. Whenever the external server evaluates a query with a condition defined on a, it returns to the requesting client the result of the query along with a *verification object* (VO). The verification object includes all the information that the client needs to know to verify the completeness of the query result (i.e., to recompute the

value of the root node) [28]. If the value of the root node computed by the user is the same as the one received from the owner, the query result is correct and complete. For instance, consider the Merkle hash tree in Figure 1.10, the verification object for a query that returns the patients whose `SSN` starts with either 2 or 3 (i.e., tuples t_2 and t_3) is represented by the gray nodes in the figure. Solutions based on authenticated data structures have the advantage of providing 100% guarantee of completeness of query results. The main problem is that these data structures can be used only for queries with conditions on the specific attribute on which they are built. This implies that the completeness of queries operating on different attributes can be checked only at the price of defining an additional authenticated data structure for each attribute.

Probabilistic approaches are based on the insertion of *fake tuples* or on the *replication* of a subset of the tuples in the outsourced/published relation. The approach based on the insertion of fake tuples [68] checks the completeness of query results by verifying whether all the fake tuples that satisfy the conditions specified in the query belong to the result computed by the server. If at least a fake tuple is missing, the query result is not complete. Clearly, fake tuples must be indistinguishable at the server's eyes from real tuples. As proved in [68], even a limited number of fake tuples ensures a high probabilistic guarantee of completeness. The solution based on the replication of a subset of the outsourced tuples [65] provides guarantee of completeness of the query result by controlling whether tuples in the query result that satisfy the condition for duplication appear twice in the query result. If a tuple that has been duplicated appears only once in the query result, the server omitted at least a tuple. Clearly, the server should not be able to determine pairs of encrypted tuples that represent the same plaintext tuple. The guarantee of completeness provided by probabilistic approaches increases with the number of additional (fake or duplicated) tuples inserted in the dataset.

Recent proposals address the problem of guaranteeing also the *freshness* of query results, meaning that queries are evaluated on the last version of the outsourced relation [45]. To integrate freshness control with solutions based on authenticated data structures, a timestamp is included in the data structure and is periodically updated [69]. If the client knows how frequently the timestamp is updated, it can check whether the verification object (and therefore the query result) returned by the server is up-to-date. The solution proposed for probabilistic approaches [68] periodically changes, in a deterministic way, the function that computes fake tuples in the dataset. If the client knows which are the current fake tuples in the outsourced data, it can verify whether the query result includes all and only those valid additional tuples that should be present when the query has been executed.

Open issues. Although the problem of providing guarantees of completeness and freshness of query results is becoming of great interest, there are still many aspects that need to be further investigated. Current solutions consider simple SELECT-FROM-WHERE SQL queries operating on one relation only. However, in many real world scenarios it is necessary to assess the correctness and completeness of the result of more complex queries (e.g., queries including GROUP BY and HAVING clauses). Probabilistic approaches provide a good trade-off between completeness guarantee and efficiency in query evaluation. It would be interesting to develop efficient approaches that provide absolute certainty of completeness of query results, while limiting the computational overhead.

1.5 Summary

In this chapter, we illustrated the main security and privacy issues arising in the emerging data outsourcing and data publishing scenarios, where a sensitive data collection is stored and managed by an external third party. For each problem analyzed, the chapter provided an overview of the most important solutions proposed in the literature and describes open issues that still need further investigation.

Acknowledgements

This work was partially supported by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4). The work of Sushil Jajodia was partially supported by the NationalScience Foundation under grant CT-20013A and by the US Air Force Office of Scientific Research under grant FA9550-09-1-0421.

1.6 Glossary

Fragmentation: Let R be a relation schema, a fragmentation of R is a set of fragments $\{F_1,\ldots,F_m\}$, where $F_i \subseteq R$, for $i=1,\ldots,m$.

Encryption: Process that transforms a piece of information (called plaintext) through an encryption algorithm to make it unintelligible.

Confidentiality constraint: Subset of attributes in a relation schema R that should not be jointly visible to unauthorized users.

Index: Piece of additional information stored at the external server together with relation R that can be used to efficiently evaluate queries operating on the attribute on which the index has been defined.

Access control policy: set of (high-level) rules according to which access control must be regulated.

Access control list (acl): Set of users who are authorized to access a given resource.

Respondent: Person (or entity) to whom the data undergoing public or semi-public release refer.

Data disclosure: Unintended release of (possibly sensitive) information that should not be revealed.

Shuffle: Process that randomly changes the assignment of node contents to physical identifiers (i.e., to the block of memory where each node is physically stored).

Digital signature: Technique used to prove that a piece of information was created by a known user and that it has not been altered.

Verification Object (VO): Piece of information sent to the client by the server executing a query, which can be used by the client to verify whether the query result is correct and complete.

Bibliography

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

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

[3] A. Ambainis. Upper bound on communication complexity of private information retrieval. In *Proc. of ICALP 1997*, Bologna, Italy, July 1997.

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

[5] R.J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of ICDE 2005*, Tokyo, Japan, April 2005.

[6] E. Bertino, P. Samarati, and S. Jajodia. Authorizations in Relational Database Management Systems. In *Proc. of CCS 1993*, Fairfax, Virginia, USA, November 1993.

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

[8] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proc. of EUROCRYPT 1999*, Prague, Czech Republic, May 1999.

[9] California senate bill sb 1386, September 2002.

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

[11] B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). In *Proc. of STOC 1997*, El Paso, Texas, USA, May 1997.

[12] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of ACM*, 45(6):965–981, April 1998.

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

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

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

[16] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-anonymity. In T. Yu and S. Jajodia, editors, *Security in Decentralized Data Management*. Springer, Berlin Heidelberg, 2007.

[17] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Microdata protection. In T. Yu and S. Jajodia, editors, *Security in Decentralized Data Management*. Springer, Berlin Heidelberg, 2007.

[18] T. Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15:429–444, 1977.

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

[20] T. K. Dang. Oblivious search and updates for outsourced tree-structured data on untrusted servers. *IJCSA*, 2(2):67 – 84, 2005.

[21] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Preserving confidentiality of security policies in data outsourcing. In *Proc. of WPES 2008*, Alexandria, VA, USA, October 2008.

[22] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Encryption-based policy enforcement for cloud storage. In *Proc. of SPCC 2010*, Genova, Italy, June 2010.

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

[24] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Private data indexes for selective access to outsourced data. In *Proc. of WPES 2011*, Chicago, IL, USA, October 2011.

[25] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Support for write privileges on outsourced data. In *Proc. of SEC 2012*, Heraklion, Crete, Greece, June 2012.

[26] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Efficient and private access to outsourced data. In *Proc. of ICDCS 2011*, Minneapolis, MN, USA, June 2011.

[27] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Supporting concurrency in private data outsourcing. In *Proc. of ESORICS 2011*, Leuven, Belgium, September 2011.

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

[29] X. Ding, Y. Yang, and R.H. Deng. Database access pattern protection without full-shuffles. *IEEE TIFS*, 6(1):189–201, March 2011.

[30] C. Dwork. Differential privacy. In *Proc. of ICALP 2006*, Venice, Italy, July 2006.

[31] V. Ganapathy, D. Thomas, T. Feder, H. Garcia-Molina, and R. Motwani. Distributing data for secure database services. In *Proc. of PAIS 2011*, Uppsala, Sweden, 2011.

[32] W. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

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

[34] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *JACM*, 43(3):431–473, May 1996.

[35] P. Golle. Revisiting the uniqueness of simple demographics in the us population. In *Proc. of WPES 2006*, Alexandria, VA, USA, October 2006.

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

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

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

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

[40] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. of the VLDB Endowment*, 3(1-2):1021–1032, September 2010.

[41] D. Kifer and A. Machanavajjhala. A rigorous and customizable framework for privacy. In *Proc. of PODS 2012*, Scottsdale, AZ, USA, 2012.

[42] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proc. of SIGMOD 2005*, Baltimore, MD, USA, June 2005.

[43] K. LeFevre, D.J. DeWitt., and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.

[44] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proc. of PODS 2010*, Indianapolis, IN, USA, June 2010.

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

[46] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In *Proc. of the ICDE 2007*, Istanbul, Turkey, April 2007.

[47] P. Lin and K.S. Candan. Hiding traversal of tree structured data from untrusted data stores. In *Proc. of WOSIS 2004*, Porto, Portugal, April 2004.

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

[49] A. Machanavajjhala, J. Gehrke, and D. Kifer. ℓ-diversity: Privacy beyond k-anonymity. In *Proc. of ICDE 2006*, Atlanta, GA, USA, April 2006.

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

[51] I. Mironov, O. Pandey, O. Reingold, and S. P. Vadhan. Computational differential privacy. In *Proc. of CRYPTO 2009*, Santa Barbara, CA, USA, August 2009.

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

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

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

[55] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. of IEEE SP 2008*, Berkeley/Oakland, CA, USA, May 2008.

[56] F. Olumofin and I. Goldberg. Privacy-preserving queries over relational databases. In *Proc. of PETS 2010*, Berlin, Germany, July 2010.

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

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

[59] Payment card industry (PCI) data security standard, September 2006. https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf.

[60] Personal data protection code. Legislative Decree no. 196, June 2003.

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

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

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

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

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

[66] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *Proc of CCS 2008*, Alexandria, VA, USA, October 2008.

[67] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE TKDE*, 23(8):1200–1214, August 2011.

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

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

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

[71] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proc. of INFOCOM 2010*, San Diego, CA, USA, March 2010.