DOI: 10.1109/CRISIS.2012.6378956

Managing and Accessing Data in the Cloud: Privacy Risks and Approaches

Sabrina De Capitani di Vimercati, Sara Foresti, Pierangela Samarati
Dipartimento di Informatica
Università degli Studi di Milano
26013 Crema - Italy
Email: *firstname.lastname*@unimi.it

***Abstract*—Ensuring proper privacy and protection of the information stored, communicated, processed, and disseminated in the cloud as well as of the users accessing such an information is one of the grand challenges of our modern society. As a matter of fact, the advancements in the Information Technology and the diffusion of novel paradigms such as data outsourcing and cloud computing, while allowing users and companies to easily access high quality applications and services, introduce novel privacy risks of improper information disclosure and dissemination.**
In this paper, we will characterize different aspects of the privacy problem in emerging scenarios. We will illustrate risks, solutions, and open problems related to ensuring privacy of users accessing services or resources in the cloud, sensitive information stored at external parties, and accesses to such an information.

***Index Terms*—Privacy risks, data protection, private access, outsourced data, cloud**

I. Introduction

Today's digital infrastructure supports innovative ways of storing, processing, and disseminating data. In fact, we can store our data in remote servers, access reliable and efficient services provided by third parties, and use computing power available at multiple locations across the network. Furthermore, the growing adoption of portable devices (e.g., PDAs, mobile phones) together with the diffusion of wireless connections in home and work environments have led to a more distributed computing scenario. These advantages come at a price of higher privacy risks and vulnerabilities as a huge amount of (private) information is being circulated and stored, often not under the direct control of its owner. In modern scenarios, like data outsourcing and cloud computing, it is hard to guarantee that sensitive data remain properly protected and that users maintain the control on who can access their data when they are stored at external cloud servers. Furthermore, the advances in the information and communication technologies, including the possibility of combining and analyzing information from several data sources, intensify the privacy problem. The proper protection of privacy requires therefore solutions for empowering users to understand the privacy risks to which they are exposed, and to maintain control over their own information, and enabling cloud servers to properly protect the privacy of their users. These problems have been under the attention of the research and development communities and several investigations have been carried out, proposing novel solutions for protecting users' privacy in data storing and publishing (e.g., [1], [2]). Research efforts have been also specifically devoted to the development of solutions addressing the privacy-aware processing of data externally stored (e.g., [3], [4]) and supporting the definition and enforcement of privacy requirements of users (e.g., [5]–[10]). Although these solutions represent important advancements in the privacy area, they tackle only part of the problem. Many issues still need to be investigated to really enable users to understand the privacy breaches to which they are exposed when interacting or exchanging information with other parties, and to maintain control over their own private data. In addition to this, solutions for efficiently accessing data while preserving their confidentiality and integrity are now complicated by the storage of data in a number of different cloud servers that need to collaborate in a private and selective way without revealing their information to other not authorized parties.

The goal of this paper is to present an overview of the main privacy issues that arise in modern scenarios. We also highlight some research directions that address the issues discussed, and describe some open problems. The remainder of the paper is organized as follows. Section II illustrates the privacy risks in cloud scenarios. Sections III-V provide a description of the main issues to be investigated for protecting the privacy of users, data stored at external cloud servers, and accesses to the externally stored data, respectively, along with some current solutions. Finally, Section VI provides our conclusions.

II. Privacy risks in the cloud

We describe the main privacy risks that arise in data outsourcing and cloud computing scenarios. Figure 1 illustrates the scenario we consider as a reference for our discussion. We assume that *data owners* store their data at external *cloud servers*, and authorized *users* (i.e., human beings or companies) access such data. The cloud servers may not be trusted for knowing the data they store and therefore data can be encrypted. For the sake of simplicity, but without loss of generality, we assume that data are organized in relational tables and are stored in a (distributed) relational database. However, the problems, solutions, and open issues illustrated in the following apply to any data model.

- *Data dissemination and sharing*. A cloud infrastructure offers several advantages to users since they can easily share and store data in a reliable system, and can have

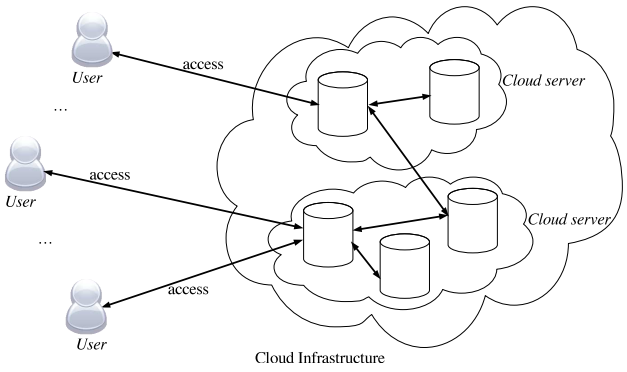

Fig. 1. Reference cloud scenario

hardware and computational power as a commodity. While this situation has an important implication from the economic point of view (the cost of hardware and the cost for managing the systems are reduced), it is creating unprecedented risks of privacy breaches. In fact, data stored and made publicly (or semi-publicly) available through cloud servers may include sensitive information that, if associated with users' identities, may violate their privacy. Several approaches adopted in a data publishing scenario to minimize the risk of unintended disclosure of sensitive information are based on k-anonymity (e.g., [11], [12]) or differential privacy (e.g., [13]). Such solutions however represent only a starting point since there are many issues that still need to be addressed. For instance, the design of techniques for protecting sensitive data from inferences that exploit dependencies among data, or the definition of a comprehensive privacy framework for formally analyzing possible information leakages.

- *Anonymous communication*. The knowledge of the fact that two parties in the cloud have exchanged messages may compromise their privacy, even when the content of the messages is kept secret. For instance, if an observer knows that a user is querying a server storing information related to cardiology diseases, the observer can infer that the user (or a person close to her) is suffering from a cardiology disease. In this case, it is necessary to protect the relationship between a user and the messages (queries) that she sends. Anonymous communication protocols (e.g., [14]) have been proposed for addressing this issue. Specific solutions taking advantage of the peculiarities of cloud systems are also being investigated (e.g., [15]).
- *Collaborative query execution*. The availability and adoption of on-demand storage and processing services require the assurance that, while releasing information and cooperating to compute query results, the confidentiality of sensitive information is not at risk. The need of a party to release information and to cooperate with another one can characterize several scenarios, ranging from traditional distributed database systems (where a centrally administered database is distributed to different locations), to cloud computing systems (where different cloud servers collaborate and integrate their data and services for providing users with applications available anywhere anytime). These situations call for innovative solutions supporting a selective sharing of information stored at multiple cloud servers, even across administrative and enterprise boundaries. In the relational database context, authorizations are usually defined through the specification of views and the definition and enforcement of access restrictions over them. Clearly, when the diversity of the parties involved and of their views is considerable and dynamic, such an approach results limiting since it requires to explicitly define a view for each possible access need. Furthermore, it imposes on the user/application the burden of knowing and directly querying the view. This aspect is particularly critical in cloud scenarios, where inter-organizational collaborations occur on a daily basis, and where the heterogeneity of the cloud servers and of their access restrictions can be high. In this case, we should devise solutions able to identify, in a declarative way, not only the portion of the data whose access is being authorized but also the visibility of possible associations that such data convey. To this purpose, some approaches take both direct and indirect information flows into consideration to determine a query plan that does not violate access restrictions specified by the owners of the different information sources (e.g., [16]–[20]).
- *External data storage*. The security and privacy concerns arising from the storage of (sensitive) information on servers outside the control of the data owners are one of the main reason for which users are often reluctant in moving to the cloud. Solutions for ensuring proper protection of the confidentiality as well as of the integrity of the data are then becoming more and more important. These problems have received the attention of the research community and several advancements have been proposed, especially in the data outsourcing context where the storing server is typically considered "honest-but-curious" (i.e., the server is trusted to properly manage and make data available to users, but it is not trusted to know the content of the data). Current solutions guarantee the confidentiality of the data by: applying a cryptographic layer on them; combining encryption with fragmentation; or departing from encryption and adopting only fragmentation (see Section IV).
  Data integrity consists in providing guarantees that cloud servers do not improperly modify the data they store without being detected. Current proposals ensure this property by adopting signature-based solutions that allow the verification of query results (see Section IV).
- *Digital interactions*. A cloud infrastructure allows users to interact with possibly unknown parties to access services and resources anywhere anytime. Such interactions may however reveal to a malicious observer (or to the

server itself) private information about the user, who may not always want to disclose her identity to gain access to the service of interest. This problem requires the adoption of appropriate techniques supporting the anonymous interaction of users with remote servers. Solutions based on *credentials* or on *anonymous credentials* can be helpful for protecting the identities of the users. Also, some researches have developed solutions allowing users to select what information to reveal during an interaction with a cloud server to gain the required access without disclosing too much personal information (see Section III).

Other important issues that arise when accessing data stored at external cloud servers are related to the protection of the query issued by a user and to the assessment of the proper behavior of the cloud server in providing a response to the query. In fact, a query may reveal sensitive information that should be kept confidential (e.g., users querying a medical database may not want to disclose to the storing cloud server which specific medical information they are interested in). Also, it is necessary to verify whether a cloud server correctly executes queries against the stored data (neither a portion of the data nor a old or modified version of them). In summary, users need guarantees that: *i)* the privacy of their queries is preserved; *ii)* the confidentiality of access patterns is not at risk (i.e., an observer should not be able to infer whether two queries aim at accessing the same data); and *iii)* the server storing the data provides a correct and complete response to their queries. The problems and solutions related to the protection of query privacy are discussed more in details in Section V.

In the following, we focus our attention on the privacy risks affecting data, users, and accesses (queries).

III. Privacy risks for users

We focus on the problem of protecting the identities of the users accessing services or resources in the cloud. In particular, we provide an overview of the main solutions for enforcing attribute-based access control, which allow users to interact with cloud servers and to release information about themselves without revealing their identities. We also describe recent approaches supporting the definition of the privacy preferences of the users. These privacy preferences specify which information is better to disclose for gaining access to a service depending on the information sensitivity.

A. Attribute-based access control

Traditional approaches for regulating access to resources are based on user authentication (e.g., [21]–[24]) and therefore cannot be adopted in the cloud, where the interacting parties can be unknown to each other. Attention has been then given to departing from user authentication and, in the name of privacy and convenience, providing access control solutions supporting credential-based and attribute-based specifications (e.g., [25]–[27]). In this way, users can easily access all the resources available from (possibly unknown) servers without the need to remember passwords or manage a specific account for each of the servers they access. In fact, credentials permit a server to verify whether the user requesting access to a service satisfies the conditions necessary to gain the access. For instance, consider a policy restricting access to a resource only to people living in a European country. In this case, the server has to communicate to the requesting user the access policy, which has to be obfuscated whenever it can be considered sensitive and as such needs to be protected [25]. After receiving the condition from the server, the user has to select and provide a credential showing where she lives, without reveling her identity. Recently, attention has been also devoted to the use of *anonymous credentials,* defining privacy-enhanced solutions and mechanisms for credential definition and management. Anonymous credentials (e.g., UProve, Idemix [28]) enable a user to selectively disclose subsets of attributes from a credential, and even to prove that the attributes contained in a credential satisfy a certain condition without revealing the exact attribute values. The integration of anonymous credentials with access control policy languages is a problem that has to be further investigated.

B. Users' privacy preferences

The release of users personal information is often regulated by approaches that can be seen as symmetric to the ones adopted by servers for regulating the disclosure of resources/services. However, access control-like specifications do not completely fit the users' protection requirements, since they may need a way to specify *preferences* on the information to disclose based on the sensitivity of such an information. For instance, a user may prefer to disclose her identity card over her passport if both credentials can allow the access to the requested service. Few proposals have addressed this issue (e.g., [5]–[10]). The proposal in [8] permits a user to associate a different cost with each credential in her portfolio representing its sensitivity (i.e., more sensitive credentials have a higher cost) and to minimize the total cost of a negotiation process. The approach in [10] proposes a point-based trust management model, where a user labels her credentials with a quantitative privacy score, and the server defines a credit for each credential that users may possess. The server allows a user access to a resource if the sum of the credits of released credentials reaches a fixed threshold. An optimal set of user's credentials is such that the total privacy score of disclosed credentials is minimal and the server's access threshold is satisfied. In [9] the authors propose a logic-based language for the specification of privacy preferences that determine a partial order among the properties of users. The approach in [5]–[7] provides a formal modeling of the user portfolio, which is composed of a set of credentials and declarations. Credentials include certified properties and are characterized by a type, a name, and the authority who issued it. Declarations are instead uncertified properties uttered by the user. Properties in the user portfolio can be either *credential-independent* when their values depend only on the credential owner (e.g., the

user name) or *credential-dependent* when their values depend on the specific credential certifying them (e.g., the credit card number). The proposed model also captures modern credential technologies (e.g., Idemix [28]) that support the selective release of arbitrary subsets of properties. These credentials are referred as *non-atomic* in contrast to traditional *atomic* credentials that imply the disclosure of all the properties they certify. Clearly, the release of a property within a non-atomic credential also entails the release of the existence of the credential certifying it. A user expresses her privacy preferences by associating a *sensitivity label* with each property and credential in the user portfolio. The sensitivity label reflects how much the user values the release of the property and of the existence of the credential. In general, the release of a set of credentials/properties entails a sensitivity corresponding to the combination of the labels of all the elements involved. There are however cases where the combined release of a set of credentials/properties is more (*sensitive view*) or less (*dependency*) sensitive than the release of the credentials/properties singularly taken. The proposed model captures these situations allowing users to associate a sensitivity label with associations of credentials and/or properties. This label specifies either the *additional* sensitivity due to the association between credentials/properties in the sensitive view, or the sensitivity that has to be *removed* since the combined release of the elements in the dependency does not provide additional information with respect to the release of the single elements. Users can also impose constraints on the disclosure of portfolio elements, by explicitly defining associations of credentials/properties that should never be released (*forbidden view*), and limitations of the form "at most n of these elements" can be jointly released (*disclosure limitation*). Figure 2(a) illustrates an example of user portfolio represented as a graph with a rectangular vertex for each credential (with label *credential_name*:type), an oval vertex for each property (with label `property_name`:value), and an edge connecting each credential to the properties it certifies. To distinguish atomic from non-atomic credentials, all the edges incident to an atomic credential are attached to a black semicircle. Sensitivity labels, which for simplicity in the example are integer numeric values, are indicated by a tag next to the vertices. The figure also represents: sensitive association {`myVISA`,`address`}, denoted with a small circle and a positive sensitivity label; dependency {`Address`,`Country`}, denoted with a small circle and a negative sensitivity label; forbidden view {`Name`,`NickName`}, denoted with a cross; and disclosure limitation {`Phone`,`eMail`}, denoted with a cross and 1 as a subscript meaning that at most one of the attributes can be disclosed.

A *minimum disclosure* is a subset of credentials and properties in the user portfolio whose release allows the user to access the service requested, while minimizing the sensitivity of the set of elements released. As an example, suppose that a user requires the access to a service that can be granted if the user releases her name and address certified by a credential of type id_card, and her name and credit card number certified by a credential of type credit_card. Figure 2(a) illustrates an example of minimum disclosure.

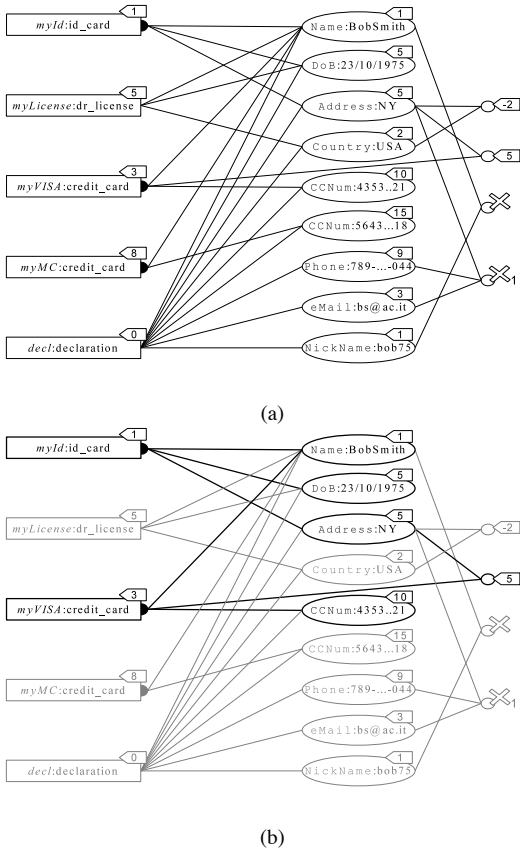

Fig. 2. An example of sensitivity specifications (a) and disclosure (b) [6]

Although this solution enables users to organize and manage all their credentials and regulate their release, there are still several open issues that need to be addressed. In fact, it is necessary to define user-intuitive approaches for expressing preferences that, for example, should derive the sensitivity labels according to some criteria (e.g., based on identity exposure), provide an integration with server-side solutions, and consider preferences varying according to the context and/or purpose of the access request.

IV. Privacy risks for stored data

Since data are stored at cloud servers that are typically not under the direct control of data owners, data confidentiality and integrity may be put at risk. Furthermore, accesses to data stored at the cloud server may need to be regulated, meaning that different parties may have different access privileges. In the following, we describe possible solutions for ensuring confidentiality and integrity of data, and for enforcing different access privileges.

A. Confidentiality and integrity

The data stored and managed by a cloud server can include sensitive information that neither the cloud server nor unauthorized users should read. The problem of protecting data confidentiality from the eyes of the storage server (as well as

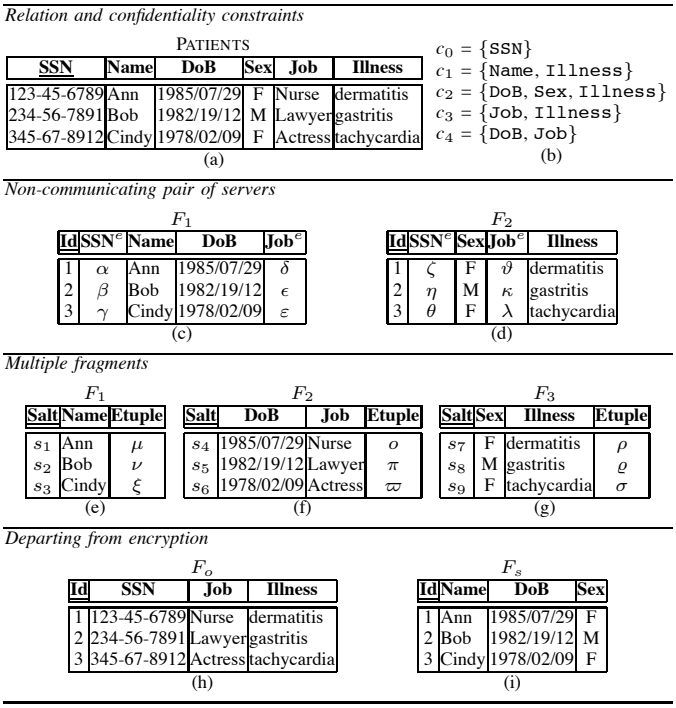

Fig. 3. An example of relation (a), a set of confidentiality constraints over it (b), and a correct fragmentation in the non-communication pairs of server (c-d), multiple fragments (e-g), and departing from encryption (h-i) scenarios

of non-authorized users accessing it) has first been addressed in the Database As a Service (DAS) scenario. The proposed solutions consist in *encrypting* the data before storing them at the external server (e.g., [2], [29]–[31]). In this way, only authorized users, who know the decryption key, can access the data content. *Indexing information* is then coupled with the encrypted data to allow the server to partially evaluate users' queries directly on the encrypted data (e.g., [29]–[31]).

Whenever data are not sensitive per se, but what is sensitive is their association with other information, encryption seems an overkill. For instance, the list of patients' names and the list of illnesses treated in a hospital can be publicly released while the association of each patient's name to a specific illness must be protected. The solutions proposed to protect the sensitive associations while limiting the adoption of encryption are based on the combined use of *encryption* and *fragmentation* (e.g., [32]–[35]). In these approaches, the sensitive associations are modeled through a set of *confidentiality constraints* [32] corresponding to sets of attributes that should not be publicly visible together. As an example, consider relation PATIENTS in Figure 3(a). A possible set of confidentiality constraints over relation PATIENTS is reported in Figure 3(b). These constraints state that the list of values of attribute SSN (c_0) is sensitive as well as the associations among: the name of each patient and the illness from which she suffers (c_1); the date of birth, sex, and illness of each patient (c_2); the job and illness of each patient (c_3); and the date of birth and job of each patient (c_4). The idea is then to protect the sensitive associations breaking them by storing data in separate relations (fragments) that cannot be joined and by possibly encrypting some attributes. The approaches relying on fragmentation and encryption to enforce confidentiality constraints can be classified as follows.

- *Non-communicating pair of servers* [32]. The data owner partitions a relation in two fragments, stored at two non-communicating servers, and encodes the attributes (i.e., attributes are transformed, for example via encryption, so that the storing servers cannot infer their original values) that cannot be stored at any of the two servers without violating confidentiality constraints. Encoded attributes are stored at both servers. Authorized users need to access both the versions of an encoded attribute to reconstruct its values. Figures 3(c-d) illustrate an example of fragmentation of relation PATIENTS in Figure 3(a) that enforces the constraints in Figure 3(b). Attribute `Id` is the tuple identifier, introduced to permit authorized users to reconstruct relation PATIENTS by joining F_1 and F_2. Attributes `SSN`^{e} and `Job`^{e} represent the encoding of attributes `SSN` and `Job`.
- *Multiple fragments* [33]. The data owner partitions a relation in an arbitrary set of disjoint fragments (i.e., each attribute appears in at most one fragment), possibly stored at the same cloud server. Each fragment stores, either in clear or in encrypted form, all the attributes of the original relation. Figures 3(e-g) illustrate an example of fragmentation of relation PATIENTS in Figure 3(a) that enforces the constraints in Figure 3(b). Attribute `Salt` is a randomly chosen value, different for each tuple in each fragment, that is concatenated with plaintext attribute values before encryption. Attribute `Etuple` is the result of the encryption of the attributes in PATIENTS that do not appear plaintext in the fragment. Note that the adoption of an arbitrary number of fragments (three in the example) permits to represent in the clear all the attributes that are not sensitive per se (attribute `Job` in the example).
- *Departing from encryption* [34]. The data owner partitions a relation in two fragments and stores one fragment locally and one fragment at an external cloud server. Only authorized users can join the two fragments. This solution completely departs from encryption for confidentiality constraint satisfaction since the data owner is willing to store a limited portion of the information. Constraints are then enforced by fragmenting the attributes in a way that sensitive associations are broken or completely stored at the data owner site. Figures 3(h-i) illustrate an example of fragmentation of relation PATIENTS in Figure 3(a) that enforces the constraints in Figure 3(b). Note that the fragment stored at the data owner side includes in the clear attributes `Job` and `Illness` forming constraint c_3, as only authorized users can access this fragment.

With respect to the integrity of the data, it is clearly not possible to prevent data tampering since the cloud server physically stores the data. However, different techniques permit the data owner and authorized users to detect unautho-

rized modifications. These solutions usually rely on *digital signatures*, possibly aggregated to minimize the overhead of the signature verification process that can detect integrity violations (e.g., [36], [37]).

Although all these solutions guarantee confidentiality and integrity protection, there are also other issues that have to be investigated. In fact, the adoption of encryption and fragmentation can compromise the utility of the data because views on them, which can be considered of interest by some recipients, are broken for privacy purposes. Also, the definition of confidentiality constraints should be extended to specify that the association among specific values of given attributes is sensitive. Finally, in a cloud infrastructure, the efficiency of data accesses is a fundamental requirement that has to be combined with the need of preserving data confidentiality.

B. Selective access

In a cloud scenario neither the data owner nor the cloud server can enforce the owner's access control policy, for confidentiality or performance reasons, respectively. In fact, the cloud server cannot directly enforce access control restrictions because it might not be trusted to enforce them and also because the policy regulating access to the data may depend on the data content (which must be kept confidential from the server). The data owner would instead need to mediate every access request to filter the query result, thus nullifying the advantages of storing data at an external server. It is therefore necessary to design a mechanism such that the data themselves enforce restrictions on the set of users who can access them. The solutions proposed are based on *selective encryption* (e.g., [38]–[41]). Selective encryption consists in encrypting different portions of the data using different keys, and in distributing keys to users in such a way that each user can decrypt all and only the pieces of information she is authorized to access. This approach is coupled with key derivation techniques for providing efficient access to the data (e.g., [42]).

V. Privacy risks for data accesses

Both the privacy of users accessing cloud services and of the data stored at cloud servers may be at risk since access requests could be exploited either by the cloud server or by a malicious observer to possibly infer the (sensitive) content of the accessed data. Also the query evaluation process may be at risk since the cloud server is not trusted and therefore can compromise the integrity of query results. We now describe in more details the main risks affecting the integrity and confidentiality of accesses evaluated by cloud servers.

A. Integrity

The cloud server (or set thereof) evaluating a query may be lazy and evaluate the user's query on a subset of the data to save computational resources, or it could execute the query on a non-up-to-date version of the data. It is therefore necessary to define a mechanism that permits users to check the integrity of query results. Query results satisfy integrity checks if they are: *correct* (i.e., computed on genuine data), *complete* (i.e., computed over the whole data collection), and *fresh* (i.e., computed on the most recent version of the data). The integrity of query results in a cloud scenario is a problem that has been addressed only recently. Correctness is usually provided by digital signature approaches. Completeness can be provided either by defining *authenticated data structures* on the data (e.g., signature chaining, Merkle hash trees, or skip lists [43]–[45]), or by inserting *sentinels* in the data collection that must also belong to query results (e.g., [46], [47]). Authenticated data structures approaches have the advantage of providing a full guarantee of query completeness, in contrast to the probabilistic guarantee offered by sentinels. However, authenticated data structures are less flexible than sentinels, since they provide integrity guarantees only for queries operating on the attribute on which the structure has been defined, and have a higher management cost. In fact, authenticated data structures cannot easily accommodate updates to the outsourced data. Freshness is provided by making authenticated data structures and/or sentinels dependent on a variable that changes over time (e.g., by inserting a timestamp updated on regular basis in the authenticated data structure or by periodically changing the function used to generate sentinels [48]).

B. Query privacy

An important issue that needs to be addressed when data are stored at external cloud servers consists in preserving the privacy of the accesses (queries) themselves. Queries can be exploited for performing different types of inferences. The first type of inference can arise in scenarios where the accessed data can be either confidential or not. As an example, consider a medical database stored at a cloud server and assume that a user accesses it looking for treatments and cures for stomach ulcer (the medical database can be encrypted or not). The user's query reveals to an observer that either the user or a person close to her suffers from stomach ulcer. The second type of inference applies when the accessed data are confidential and are encrypted. Queries can be exploited for inferring information about the data content. In this case, it is not sufficient to protect the confidentiality of queries singularly taken (i.e., *access confidentiality*), but it is also necessary to protect the confidentiality of patterns of accesses (i.e., *pattern confidentiality*), meaning that the server should not be able to infer whether two different queries aimed at the same target information. In fact, by monitoring accesses to the data, the cloud server could exploit the information on the frequencies with which different pieces of information are accessed to reconstruct the correspondence between the plaintext data and the encrypted data.

Solutions proposed for protecting access and pattern confidentiality when data are stored in the clear are typically based on Private Information Retrieval (PIR) techniques and provide access and pattern confidentiality at a high computational and communication cost (e.g., [49]). Proposals operating on confidential data introduce privacy-preserving indexing techniques (e.g., based on Oblivious RAM and B-tree data structures)

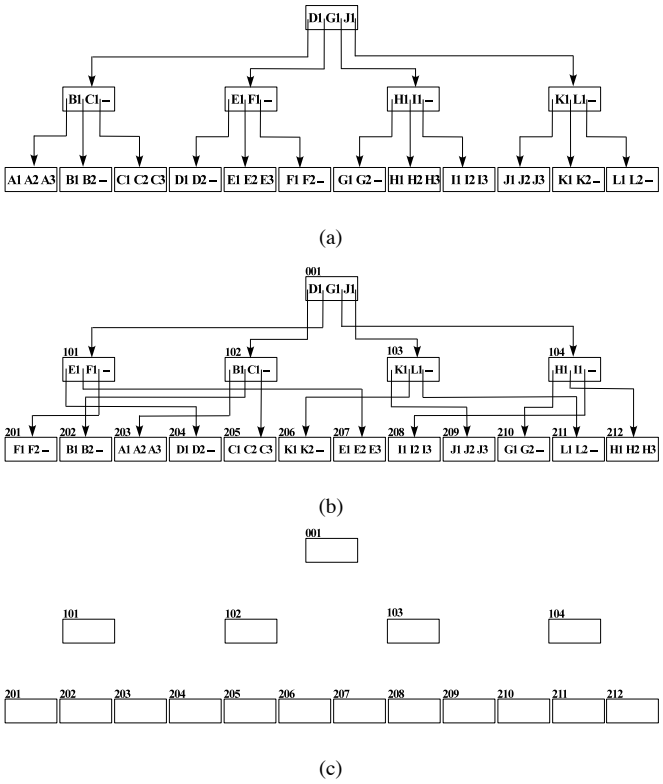

Fig. 4. An example of abstract (a), logical (b), and physical (c) representation of a shuffle index

able to compute the query result while providing guarantees of access and pattern confidentiality (e.g., [3], [4]). One of these solutions consists in the definition of a *shuffle index* for data organization and storage [3] (which also supports concurrent accesses to the data [50]). The data stored at the cloud server are indexed over a candidate key K and are organized as an *unchained* $B+$*-tree* with fan-out F (i.e., each node except the root has $q \geq \lceil F/2 \rceil$ children and stores $q-1$ values $v_1 \leq \ldots \leq v_{q-1}$). Data are stored in the leaves of the unchained $B+$-tree in association with their index values. The reason for the absence of the links between leaves is that if such links are followed during a search operation, they disclose the order relationship among the index values. Figure 4(a) illustrates an example of abstract unchained $B+$-tree.

At the logical level, each node n in the unchained $B+$-tree is associated with a *logical identifier id*, and is represented by the pair $\langle id, n \rangle$, with n the node content. Pointers between nodes at the abstract level translate to logical identifiers at the logical level. The order between node identifiers does not necessarily reflect the value-order relationship in the abstract nodes. Figure 4(b) illustrates a possible logical representation of the abstract unchained $B+$-tree in Figure 4(a). For simplicity, in the figure nodes appear ordered according to their logical identifier (reported on the top of each node) whose first digit corresponds to the level of the node in the tree.

At the physical level, the encryption of a node content is obtained concatenating it with a random salt to destroy plaintext distinguishability and applying symmetric encryption in CBC mode. The logical identifier of the node easily translates into the physical address where the block representing the node is stored at the server (for simplicity, we assume that the physical address of a block coincides with the logical identifier of the corresponding node). Figure 4(c) illustrates the physical representation of the logical index in Figure 4(b), which corresponds to the view of the server over the data collection it stores.

Although encryption provides confidentiality of data at rest and access confidentiality of individual requests, it is not sufficient for ensuring pattern confidentiality. In fact, if the server observes two accesses that retrieve the same block, it can immediately infer that they aimed at the same node content and, by observing a long enough history of accesses, it can reconstruct the frequency of accesses to blocks. This information, combined with the information about the frequency with which different values are accessed, allows the server to reconstruct the node-block correspondence (frequently accessed blocks store frequency accessed nodes). To avoid these problems, encryption is complemented with *cover searches*, *cached searches*, and *shuffling*.

- *Cover searches*. Cover searches are *fake* searches executed in conjunction with the actual *target* search. Cover searches aim at introducing confusion on the target of a search operation: hiding the actual search within a set of fake searches, any of the leaf blocks returned by the searches has the same probability of storing the target value. To provide such a protection guarantee, cover searches must be *indistinguishable* from actual searches (at the cloud server's eyes), and provide *block diversity* (i.e., they must follow disjoint paths in the shuffle index except for the root). Adopting cover searches, each access execution requires to download from the server *num_cover* $+1$ blocks for each level of the shuffle index, where: one block stores the node along the path to the target value, and the other *num_cover* blocks store the nodes along the paths to the *num_cover* cover searches.
- *Cached searches*. The cache is a layered structure, with *num_cache* elements for each layer, that stores at the client side a *plaintext copy* of the last *num_cache* visited paths to target nodes. The cache keeps track of actual (not of cover) searches and is managed according to the Least Recently Used policy, which guarantees that the parent of any node in cache is in the cache as well as the path from the root of the shuffle index to any node in the cache (path continuity property). The cache aims at protecting the accessed data against intersection attacks, by making searches for the same target (within a short time interval) indistinguishable from searches for different target values. If the target already belongs to the local cache, *num_cover* $+1$ covers are used in the access.
- *Shuffling*. Most inferences that the server can draw are caused by the one-to-one correspondence existing between a node and the block storing it. Node shuffling aims at breaking this correspondence by *moving* the content

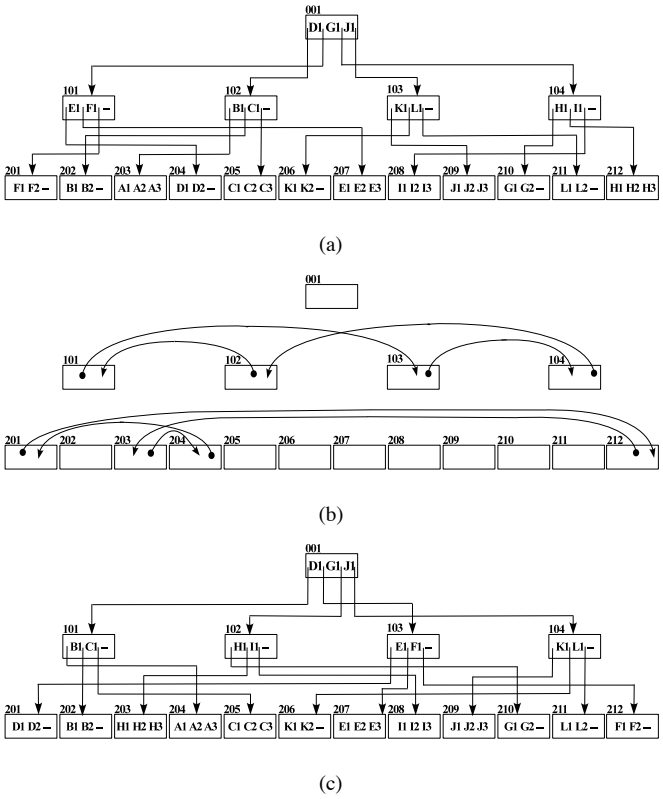

Fig. 5. An example of evolution of the logical shuffle index in Figure 4(b) as a consequence of a search for value 'D2' with 'H3' and 'L1' as covers

of accessed (either as target or as covers) and cached nodes to different blocks. Shuffling modifies the node-block allocation, and implies a re-encryption of the shuffled nodes, thus changing their encrypted representation. This prevents the server from reconstructing the shuffle operation by exploiting the encrypted block content. The parents of shuffled nodes need to be updated to preserve the correctness of the index structure. Shuffling prevents the server from inferring whether two accesses to the same block aimed at the same target node content.

We now describe how a search operation can be executed through the combination of the three techniques described above. Given the target *target_value* of the search, the client first chooses *num_cover* $+ 1$ cover searches. The search then requires an iterative process between the client and the server. For each level l of the shuffle index, the client downloads from the server *num_cover* $+ 1$ blocks: one representing the node along the path to *target_value* (if not in cache); and *num_cover* (*num_cover* $+ 1$ if the node along the path to *target_value* is in cache) along the paths to cover searches. The client decrypts the blocks downloaded from the server and shuffles their content with cached nodes at level l. To maintain the correctness of the shuffle index structure, it updates the pointers to children in the parents of shuffled nodes. The client then updates the local cache at level l either inserting the node at level l along the path to *target_value* (if such a node was not in cache before the search), or making the node along the path to *target_value* the least recently used (if the node was in cache before the search). The client then encrypts the nodes at level $l - 1$, either downloaded from the server or in cache, and sends the resulting blocks to the server for storage. The process ends when it reaches the leaves of the shuffle index. The leaf along the path to *target_value* is returned to the user since it contains the tuple with value *target_value* for attribute K (if such a tuple exists in the dataset). For instance, consider a search for 'D2' in the shuffle index in Figure 5(a) and assume that *num_cover*=1, *num_cache*=2, and that the cache stores nodes $001\,[_{102}\mathrm{D1}_{101}\mathrm{J1}_{104}\mathrm{G1}_{103}]$; $101\,[_{204}\mathrm{E1}_{207}\mathrm{F1}_{201}-]$; $102\,[_{203}\mathrm{B1}_{202}\mathrm{C1}_{205}-]$; $201\,[\mathrm{F1F2}-]$; $203\,[\mathrm{A1A2A3}]$. The client chooses two covers for 'D2' (e.g., 'H3' and 'L1') and visits the root node (i.e., block 001) in cache to determine the blocks at the first level to download from the server. Since block 101 along the path to 'D2' is already in cache, the client downloads and decrypts blocks 104 and 103 (along the paths to 'H3' and 'L1', resp.). The client shuffles nodes 101, 102, 103, and 104 as illustrated in Figure 5(b), updates the root node (and its representation in cache) accordingly, encrypts its content and sends the resulting block to the server for storage. The client then determines the blocks at the leaf level that must be downloaded: since block 204 along the path to 'D2' is not in cache, it discards one of the two cover searches (e.g., 'L1') and downloads blocks 204 and 212 (along the paths to 'D3' and 'H3', resp.). The client shuffles nodes 201, 203, 204, and 212 as illustrated in Figure 5(b) and updates nodes 101, 102, 103, and 104 (and their representation in cache) accordingly. The client encrypts nodes 101, 102, 103, and 104 and sends the resulting blocks to the server. It then updates the cache at the leaf level inserting $201\,[\mathrm{D1D2}-]$ and removing $204\,[\mathrm{A1A2A3}]$. Finally, the client encrypts leaf nodes 201, 203, 204, and 212 and sends the blocks to the server. Figure 5(c) illustrates the logical shuffle index after the access.

The solutions based on the definition of privacy-preserving indexing techniques successfully provide effective protection to access and pattern confidentiality. There are however different open issues that need to be further investigated, to make these access technique flexible enough to suit the requirements of any data collection. In fact, these approaches could be extended to support the definition of multiple indexes on the same outsourced data, updates to the data collection (i.e., insert, delete, and update operations), and to support not only equality queries but also queries including range conditions, grouping operations, aggregate functions, and so on.

VI. Conclusions

Effective and efficient solutions for protecting the privacy of the parties interacting in a cloud infrastructure as well as of the data stored and processed are of paramount importance for enabling a widespread exploitation of the cloud technology. Privacy is however far from being a trivial problem to address and represents a big challenge for all parties that use and develop cloud technology. In this paper, we discussed the main privacy risks that arise in a cloud scenario and illustrated some solutions for addressing them.

ACKNOWLEDGMENTS

The paper is based on joint work with C.A. Ardagna, S. Jajodia, G. Pelosi, and S. Paraboschi. This work was supported in part by a Google Faculty Research Award on "Fine-Grained Access Control for Social Networking Applications".

REFERENCES

[1] B. Fung, K. Wang, R. Chen, and P. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM CSUR*, vol. 42, no. 4, pp. 14:1–14:53, Jun. 2010.

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

[3] 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*, Minneapolis, MN, USA, Jun. 2011.

[4] 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*, Alexandria, VA, USA, Oct. 2008.

[5] C. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati, "Minimizing disclosure of private information in credential-based interactions: A graph-based approach," in *Proc. of PASSAT*, Minneapolis, MN, USA, Aug. 2010.

[6] ——, "Minimising disclosure of client information in credential-based interactions," *IJIPSI*, vol. 1, no. 2/3, pp. 205–233, 2012.

[7] ——, "Supporting privacy preferences in credential-based interactions," in *Proc. of WPES*, Chicago, IL, USA, Oct. 2010.

[8] W. Chen, L. Clarke, J. Kurose, and D. Towsley, "Optimizing cost-sensitive trust-negotiation protocols," in *Proc. of INFOCOM*, Miami, FL, USA, Mar. 2005.

[9] P. Kärger, D. Olmedilla, and W.-T. Balke, "Exploiting preferences for minimal credential disclosure in policy-driven trust negotiations," in *Proc. of SDM*, Auckland, New Zealand, Aug. 2008.

[10] D. Yao, K. Frikken, M. Atallah, and R. Tamassia, "Private information: To reveal or not to reveal," *ACM TISSEC*, vol. 12, no. 1, pp. 1–27, Oct. 2008.

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

[12] P. Samarati, "Protecting respondents' identities in microdata release," *IEEE TKDE*, vol. 13, no. 6, pp. 1010–1027, Nov. 2001.

[13] C. Dwork, "Differential privacy," in *Proc. of ICALP*, Venice, Italy, Jul. 2006.

[14] C. Ardagna, S. Jajodia, P. Samarati, and A. Stavrou, "Providing mobile users' anonymity in hybrid networks," in *Proc. of ESORICS*, Athens, Greece, Sep. 2010.

[15] N. Jones, M. Arye, J. Cesareo, and M. Freedman, "Hiding amongst the clouds: A proposal for cloud-based onion routing," in *Proc. of FOCI*, San Francisco, CA, USA, Aug. 2011.

[16] M. Benedikt, P. Bourhis, and C. Ley, "Querying schemas with access restrictions," *Proc. of VLDB Endowment*, vol. 5, no. 7, pp. 634–645, Mar. 2012.

[17] A. Calì and D. Martinenghi, "Querying data under access limitations," in *Proc. of ICDE*, Cancun, Mexico, Apr. 2008.

[18] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Assessing query privileges via safe and efficient permission composition," in *Proc. of CCS*, Alexandria, VA, USA, Oct. 2008.

[19] ——, "Controlled information sharing in collaborative distributed query processing," in *Proc. of ICDCS*, Beijing, China, Jun. 2008.

[20] C. Li, "Computing complete answers to queries in the presence of limited access patterns," *VLDB Journal*, vol. 12, no. 3, pp. 211–227, Oct. 2003.

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

[22] ——, "Privacy in biometrics," in *Biometrics: theory, methods, and applications*, N. Boulgouris, K. Plataniotis, and E. Micheli-Tzanakou, Eds. Wiley-IEEE Press, 2009.

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

[24] R. Sandhu and P. Samarati, "Authentication, access control and intrusion detection," in *CRC Handbook of Computer Science and Engineering*, A. Tucker, Ed. CRC Press Inc., 1997, pp. 1929–1948.

[25] C. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, and M. Verdicchio, "Expressive and deployable access control in open web service applications," *IEEE TSC*, vol. 4, no. 2, pp. 96–109, Apr-Jun. 2011.

[26] P. Bonatti and P. Samarati, "A uniform framework for regulating service access and information release on the Web," *JCS*, vol. 10, no. 3, pp. 241–272, 2002.

[27] A. Lee, M. Winslett, J. Basney, and V. Welch, "The Traust authorization service," *ACM TISSEC*, vol. 11, no. 1, pp. 1–3, Feb. 2008.

[28] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Proc. of EUROCRYPT*, Innsbruck, Austria, May 2001.

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

[30] 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*, Washington, DC, USA, Oct. 2003.

[31] H. Wang and L. Lakshmanan, "Efficient secure query evaluation over encrypted XML databases," in *Proc. of VLDB*, Seoul, Korea, Sep. 2006.

[32] 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*, Asilomar, CA, USA, Jan. 2005.

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

[34] ——, "Keep a few: Outsourcing data while maintaining confidentiality," in *Proc. of ESORICS*, Saint Malo, France, Sep. 2009.

[35] ——, "Fragmentation design for efficient query execution over sensitive distributed databases," in *Proc. of ICDCS*, Montreal, Quebec, Canada, Jun. 2009.

[36] 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.

[37] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM TOS*, vol. 2, no. 2, pp. 107–138, May 2006.

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

[39] ——, "A data outsourcing architecture combining cryptography and access control," in *Proc. of CSAW*, Fairfax, VA, USA, Nov. 2007.

[40] ——, "Encryption policies for regulating access to outsourced data," *ACM TODS*, vol. 35, no. 2, pp. 12:1–12:46, Apr. 2010.

[41] ——, "Support for write privileges on outsourced data," in *Proc. of SEC*, Heraklion, Crete, Greece, Jun. 2012.

[42] M. Atallah, M. Blanton, N. Fazio, and K. Frikken, "Dynamic and efficient key management for access hierarchies," *ACM TISSEC*, vol. 12, no. 3, pp. 18:1–18:43, Jan. 2009.

[43] M. Narasimha and G. Tsudik, "DSAC: Integrity for outsourced databases with signature aggregation and chaining," in *Proc. of CIKM*, Bremen, Germany, Oct.–Nov. 2005.

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

[45] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated join processing in outsourced databases," in *Proc. of SIGMOD*, Providence, RI, USA, Jun.-Jul. 2009.

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

[47] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *Proc. of VLDB*, Vienna, Austria, Sep. 2007.

[48] ——, "Providing freshness guarantees for outsourced databases," in *Proc. of EDBT*, Nantes, France, Mar. 2008.

[49] W. Gasarch, "A survey on private information retrieval," *Bulletin of the EATCS*, vol. 82, pp. 72–107, 2004.

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