Data Security and Privacy in the Cloud

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

Dipartimento di Informatica, Università degli Studi di Milano
firstname.lastname@unimi.it

ABSTRACT

Relying on the cloud for storing data and performing computations has become a popular solution in today's society, which demands large data collections and/or analysis over them to be readily available, for example, to make knowledge-based decisions. While bringing undeniable benefits to both data owners and end users accessing the outsourced data, moving to the cloud raises a number of issues, ranging from choosing the most suitable cloud provider for outsourcing to effectively protecting data and computation results. In this paper, we discuss the main issues related to data protection arising when data and/or computations over them are moved to the cloud. We also illustrate possible solutions and approaches for addressing such issues.

Keywords: Data security, privacy, cloud computing, emerging scenarios

1. INTRODUCTION

We live in a world that more and more relies on the analysis of large data collections to create knowledge and generate knowledge-based decisions. It is no surprise that data are regarded by many as the oil of the digital era and, thanks to the advancements of the Information and Communication Technologies (ICTs) and the availability of new technologies such as the Internet-of-Things (IoT), the availability of huge collections of data is increasing at a fast pace. Cloud computing is accompanying this trend as an enabling factor: as a simple yet enlightening example, it is already well-known that the reliability, availability, flexibility, cost savings, and security of the cloud are essential for the development of machine-to-machine (M2M) services in the IoT, and will be more so in the future. More in general, thanks to the elastic storage and computation capabilities that cloud providers can offer to customers at competitive prices, the cloud represents a key computational model for storing, analyzing, and sharing resources or large collections of data.[1] The general reference scenario is characterized by data owners, who rely on the platforms and solutions made available by cloud providers for storing data, and external users, who need access to those data and possibly delegate computations over them to the cloud.

While the cloud can certainly bring to users and owners uncountable benefits, moving to the cloud for data storage and analysis also entails a number of problems and issues that need to be carefully addressed.[2] One of the problems that need to be addressed derives form the availability, in the current cloud market, of a large number of providers offering an impressive number of solutions that can be rented/purchased, tailored to meet different application scenarios and requirements. It comes without saying that choosing one solution over another can be the key for successfully meeting the specific requirements and needs of both the owner of the data to be outsourced, as well as of the users accessing and/or analyzing data. Another problem derives form the fact that (portions of) the data that an owner may wish to outsource to the cloud can be sensitive, and hence should be protected. If the cloud provider cannot be considered fully trusted, it is then necessary to have models and techniques for adequately protecting the outsourced data in terms of both confidentiality (meaning that unauthorized parties, possibly including the provider itself, cannot access the sensitive data content) and integrity (meaning that unauthorized modifications or deletions of data can be detected and discarded/counteracted). A further problem derives from the fact that outsourced data can need to be shared, possible selectively, with other users, and that analysis (e.g., queries) over them may need to be executed. It is then necessary to have models and techniques that allow for selective and fine-grained data retrieval, and for ensuring that the results of computations are correct and complete. Strictly related to this issue, there is the problem that sometimes the queries themselves can be considered sensitive and should be protected from unauthorized parties: for example, a provider observing that a certain user regularly queries a medical dataset for retrieving the treatments for a specific cardiovascular disease might infer that the user, or an individual close to her, can suffer from a heart-related illness. It is then

necessary to have means for keeping queries confidential. A last aspect that can arise relates to the possibility of leveraging the diversity and richness of the cloud market for collaborative computations: by allocating different operations to different players in the market, it is possible to obtain significant economic savings.

In this paper, we discuss the issues above and possible approaches and solutions to them. The paper includes a section for each of the issues addressed: selection of cloud plans (Section 2); protection of the confidentiality and integrity of data in storage (Section 3); enforcement of selective and fine-grained access to data (Section 4); protection of the confidentiality of queries and assessment of the integrity of their results (Section 5); and collaborative execution of queries (Section 6). We finally conclude the paper in Section 7.

2. CLOUD PLAN SELECTION

The current cloud market is characterized by a large number of service providers, which offer their solutions in terms of pre-defined configurations (i.e., *plans*) with different characteristics that make, for example, a plan (e.g., with strong security guarantees) more suitable for storing sensitive data, and another (e.g., with strong guarantees on integrity and performance) for storing mission-critical but non-sensitive data. A key problem for a user wishing to delegate data storage and management to cloud-based platforms is related to the selection of the most suitable plan(s) for outsourcing. In this section, we discuss the problem of cloud plan selection and possible approaches for it.

Problem. While clearly the more plans available in the market, the more each user can find a good fit for her needs, the selection of a plan can raise a number of problems that need accurate solutions. First, the user must determine which are the parameters of interest for her choice (and hence, the characteristics over which evaluating available plans). A straightforward solution could be to rely on the Service Level Agreement (SLA) specifications that each cloud provider publishes for her plans and pick the parameters of interest (e.g., those related to security/performance if the dataset to be stored is sensitive/mission-critical). While this might seem trivial, it is not – different SLAs can include different information, or it may also happen that the same information is represented with different terminologies. Once the parameters of interest have been identified, a second problem relates to supporting users in specifying, in an easy and intuitive way, their requirements, which may be rich and complex. Relying on pre-defined baselines and requirements (such as the requirements for an affordable plan with high performance) might not be a good option, because different users might have different and possibly contrasting needs to be considered. Hence, there is a need for supporting users in specifying in an easy and effective way arbitrary and rich requirements.[3] Note that assuming users to specify requirements on the values of the service parameters specified in SLAs can be limiting, for different aspects. For instance, the uptime of plans is typically expressed in SLAs with a percentage. A user who needs a performant plan for mission-critical data might set a threshold to the uptime values, requiring a monthly uptime greater than 99.99% and hence discarding all those plans that do not guarantee such a lower bound. Indeed, operating over crisp values implicitly introduces sharp boundaries between values that are acceptable (e.g., 99.99% uptime) and values that are not (e.g., a plan with 99.989% uptime would be discarded, although such a close value could as well as be bearable). Also, the features that characterize cloud service plans can be technically low-level and not immediately understandable to non-skilled users. Hence, there is a need for departing from the crisp values and pre-defined terminologies used in SLAs, to ensure that all users are able to specify their requirements. A last problem, arising after parameters of interest have been captured and expressive requirements have been formulated, is related to the selection process, which requires to determine a precise method for assessing a certain plan: also in this case the problem is complex as there might be different, and possibly contrasting, criteria (deriving from user requirements) that should be taken into consideration in the assessment phase.

Solutions. The existing solutions that address the problems illustrated above typically rely on the introduction of a broker, which is an entity mediating the interactions between users and providers to the aim of evaluating users' requirements and mapping them onto the characteristics of the offered plans.[4,5]

The solutions proposed for the definition of the parameters to be used for the formulation of requirements operate with parameters ranging from a simple cost/performance ratio guaranteed by a plan,[6] to sets of (pre-defined) QoS attributes declared by a provider for service provision,[4] to security-related attributes and properties.[7–10]

The first attempts for the specification of complex requirements only supported lower-bounds to QoS values.[4] More recent proposals have been specifically designed to provide users with easy-to-use yet expressive and rich languages and models for the specification of requirements. A recent approach[3] proposes a high-level and user-friendly language for specifying requirements and preferences that a user may want to define. Requirements must be satisfied by a plan for being considered acceptable. For instance, '`provider` IN {Ghost,Amaron}' is an example of a base requirement that demands that the provider of a plan be Ghost or Amaron, and hence a plan that is managed by a provider different from those is to be discarded. The language for requirements specification allows users to define, besides base requirements, also alternatives and conjunctions among them, conditional (if-then) requirements, and forbidden configurations of parameters. Requirements can be complemented with preferences, which correspond to soft constraints that are evaluated against those plans that satisfy the requirements to determine which one(s) is (are) considered more satisfactory (i.e., preferences are used to rank the plans). While this proposal provides an expressive and friendly language for users, it still requires some understanding of the (possibly low-level) characteristics of the cloud plans for the specification of requirements and preferences. To provide support also to non-skilled users, a more flexible approach[11] makes a further step and allows for the specification of requirements for cloud plan selection using natural language expressions (e.g., terms such as 'high' and 'low', instead of crisp values). The requirements can be expressed as fuzzy parameters, associating natural language expressions to low-level technical characteristics of cloud plans (e.g., '`throughput` IS MEDIUM'), and fuzzy concepts, associating natural language expressions to higher-level abstractions easily understandable and usable by non-skilled users (e.g., '`security` IS HIGH').[12] As for the models and techniques that can be used to enforce user requirements and assess cloud plans, many approaches have addressed the possibility of accommodating multiple/contrasting requirements and reconciling requirements coming from multiple applications, for instance, through multi-criteria decision making techniques.[4,13] Other approaches have focused on considering also feedbacks from personal experience,[14] QoS values prediction,[15] the management of dependencies existing among requirements and/or plans characteristics,[16–18] and fuzzy logic and fuzzy inferences.[11]

3. PROTECTION OF DATA AT REST

Delegating to a cloud provider the storage and management of a data collection raises concerns about the security of such data since, upon relying on external platforms, the data owner loses her physical control over her own data. In this section, we discuss two key issues that need to be addressed when moving data to the cloud: protecting the confidentiality of the data (Section 3.1), and assessing their integrity (Section 3.2).

3.1 Data confidentiality

The need of ensuring data confidentiality in the cloud context is motivated by the fact that data may be sensitive or confidential and therefore should not be disclosed to unauthorized subjects. In the following, we discuss the problem of ensuring that an outsourced data collection is maintained confidential, and possible approaches for it.

Problem. Data that need to be stored in the cloud can be sensitive. If this is the case, it is necessary to ensure that the cloud provider storing the data cannot access the (plaintext) values of the data. In fact, a common safe assumption is that cloud providers be *honest-but-curious*, that is, trusted for providing services and managing data, but not for accessing their contents.

Solutions. A traditional solution for ensuring confidentiality consists in protecting the data through encryption. In principle, encryption can be *server-side* (i.e., when data are encrypted by the cloud provider upon data reception) or *owner-side* (i.e., when data are encrypted by the owner before being sent to the provider). Server-side encryption clearly grants the cloud provider full visibility on the data it stores, and hence implicitly requires that the owner fully trusts the provider also for data confidentiality. Owner-side encryption, on the contrary, maintains data confidential also to the provider itself, keeping the owner in control of her data, and hence represents an effective solution for the confidentiality of data in the presence of honest-but-curious providers.[19] Indeed, the downside of owner-side encryption is that since decryption can be performed at the data owner side only, the functionality that can be offered by the provider on the data are limited. For instance (as will be illustrated in more details in Section 4), encryption makes query execution difficult,[20] and solutions have been developed for

supporting SQL queries over encrypted data[21] as well as indexes for query execution.[22]
To limit the use of encryption, a different approach for protecting data confidentiality is based on *data fragmentation*.[19] Intuitively, approaches based on data fragmentations operate by splitting the original data collection in a certain number of unlinkable *fragments* (i.e., views in the context of relational databases). Fragmentation can be effectively adopted whenever data are not sensitive per se, but what is sensitive is the *association* among data items. For instance, consider a medical dataset. While the associations among a set of patients and their diseases is certainly a sensitive piece of information that should be protected, the list of patients and the list of diseases singularly taken might not be considered sensitive. In this case, encrypting the entire dataset might be an overdo, and the protection of the sensitive associations can be enforced by splitting the dataset in two (or more) fragments such that the names of the patients and their diseases be stored in different fragments. Existing fragmentation-based approaches, developed in the context of relational databases, differ in how and whether fragmentation is coupled with encryption. Indeed, when a data item (e.g., an attribute of a relational table) is sensitive per se, it cannot be fragmented and hence it has to be encrypted. The first fragmentation-based approach[23] assumes the existence of two non-communicating providers, and fragments a relation in two fragments only, each stored at one provider. Fragments have a common attribute that can be used for join, and hence the two providers are assumed not to collude. Encryption is adopted for protecting sensitive data and those associations that cannot be broken by fragmentation. To overcome the assumption of non-collusion among the two providers, other approaches have been proposed, producing fragmentations that can be composed of an arbitrary number of fragments that could as well be stored at a single provider.[24] Protection in this case is ensured by the fact that the fragments do not have common attributes, and hence cannot be joined. To allow execution of queries, each fragment also includes in encrypted form (hence only available to authorized subjects) all attributes that should not be visible in the fragment. A possible risk with this approach is represented by the existence of dependencies among data, which might allow a (loose) recombination of some of the fragments. As an example, a fragment storing patients' diseases and a fragment storing patients' physicians might be joined thanks to the link between a physician and the illnesses she treats. A specific approach has then been developed for producing fragmentations that can be stored at a single provider without the risks due to data dependencies.[25]
To completely depart from encryption, a possible approach requires the data owner to store at her premises a limited portion of the data,[26] in such a way to protect those data that could not be protected by fragmentation.

3.2 Data integrity and availability

Besides protecting the confidentiality of the data stored in the cloud, other key problems concern guaranteeing data integrity, authenticity, and availability. We now discuss these issues and possible approaches for them.

Problem. Ensuring data integrity means that data stored in the cloud should be protected against improper (i.e., malicious and/or accidental) changes. This is particularly important whenever cloud providers are not fully trustworthy and their possibly lazy or malicious behavior should be controlled. Data availability in the cloud mainly refers to verifying that data are managed by the storing provider according to its SLA.

Solutions. Classical solutions for ensuring data integrity are based on Message Authentication Codes (MACs) or digital signatures.[27–30] These approaches require the availability of a pair of asymmetric keys (for digital signatures) or of a symmetric key (for MACs). When a user wants to verify the integrity of the data, she has to first download the data as well as the corresponding signature or MAC from the cloud, and then check whether the signature/MAC matches the one she computes on the downloaded data. If there is a match, then the data was not modified while stored in the cloud. While effective, such solutions are not applicable whenever the amount of data to be checked is considerable, due to the overhead caused by the need for downloading all data as well as its cost (the out-bound traffic usually has a cost proportional to the amount of transferred data).
To avoid the download of the entire data collection, alternative (probabilistic) solutions have been proposed, such as Provable Data Possession (PDP)[31] and Proof Of Retrievability (POR).[32] PDP is based on the pre-computation of *homomorphic verification tags* that are stored together with the corresponding blocks of the file outsourced to an external server. A user (including the data owner) aiming to verity the integrity of her data (i.e., to check whether the server stores all blocks composing her file) has to first generate a random challenge against a randomly selected set of file blocks. The server uses the selected blocks and the corresponding tags for generating a proof of possession (note that, thanks to the homomorphic property of the tags, they can

be combined into a single value, thus obtaining compact proofs). POR can be applied only on encoded (e.g., encrypted) resources stored at an external server. The basic idea consists in encrypting the file and inserting *sentinels*, indistinguishable from the original blocks, in the outsourced encrypted file. When the data owner wishes to verity the integrity of the file, she specifies the positions of a set of sentinels and asks the server to return the corresponding values. If the server has deleted or modified the original file, with high probability, such change also affects the sentinels, meaning that it cannot correctly respond to the data owner. Starting from the two basic PDP and POR protocols described above, further improvements and variations have been then proposed.[33–35] Similar approaches, building on the insertion in the outsourced dataset of ad-hoc fake and replicated data (e.g., twins and markers[36,37]), known to subjects authorized to verify data integrity but indistinguishable to real data to the eyes of the cloud provider, have been developed to verify both the integrity of an outsourced data collection and of the results of a query (as will be illustrated in more details in Section 5). Alternative solutions are based on the adoption of an independent *third-party auditor* (TPA) that periodically verifies the integrity of outsourced data.[38]
Data availability is typically guaranteed by replicating the data in different servers and by running distributed storage protocols to mask failures of single servers.[39,40]

4. SELECTIVE AND FINE-GRAINED ACCESS TO DATA

The fact that cloud providers are not typically considered fully trusted to access the content of the data they store can have an impact whenever the outsourced data are to be retrieved. In this section, we discuss the two main problems that can arise: the execution of queries, when the outsourced data are encrypted or fragmented (Section 4.1), and the enforcement of authorization policies, when the provider is not trusted to act as reference monitor (Section 4.2).

4.1 Fine-grained access

While protecting data confidentiality, owner-side encryption and fragmentation (Section 3) prevent fine-grained access to data, as the provider cannot bypass the enforced protection. In the following, we discuss the problem of enforcing fine-grained access to data, and possible approaches for it.

Problem. Since cloud providers are not authorized to access the content of the data they store, they cannot easily evaluate a query that needs to retrieve portions of the data that satisfy the conditions expressed in the clauses of a query. If the dataset is encrypted, the provider cannot access the plaintext values to evaluate the conditions since, for protecting confidentiality, encryption keys are known to the data owner and to authorized users only. If the dataset is fragmented, the provider cannot reconstruct sensitive associations (broken by fragmentation) between the tuples in the fragments.

Solutions. When a dataset is encrypted to protect its confidentiality, the data content is hidden to the provider and hence query evaluation cannot be delegated. Query evaluation then requires the requesting user to download the entire outsourced dataset and to evaluate the query locally. This is however not a viable solution, as the overhead introduced would clearly diminish the advantage of resorting to cloud platforms for data storage. A possible solution for (partially) delegating to the cloud provider the evaluation of queries over encrypted data relies on the definition of *indexes*. Originally proposed in the context of relational databases, indexes are metadata that reflect some of the properties of the values of the original attributes over which they are defined, without disclosing sensitive values.[19] Indexes are represented as additional attributes added to the original relation schema: if an attribute of the original relation schema is expected to be involved in query evaluation, then an index is defined for it. Query evaluation on an encrypted and indexed relation requires the original query (formulated on the original schema) to be rewritten in two queries: one operating at the provider side on indexes, and one operating at the user side on the result computed over indexes (to remove the spurious results that the imprecise evaluation caused by indexes can introduce). The definition of indexes must take into consideration the type of condition (e.g., equality, range, aggregate operations) that is expected to be evaluated.[20,41,42] An alternative solution to the definition of indexes is based on the adoption of encryption schemes that support the search of keywords[43] or of homomorphic encryption[44] to permit the evaluation of conditions directly over encrypted data. Different encryption schemes (e.g., order-preserving, deterministic, random) can also be used in

combination, in an onion-like fashion: in this case, each data item is wrapped in multiple layers of encryption, where each layer allows for the evaluation of a specific condition/operation.[45] Layers can be peeled off by authorized users and are organized by decreasing protection guarantees.
When the outsourced relation is fragmented, rather than encrypted, a (partial) support for query execution can be provided by the definition of *loose associations* over the fragments.[46,47] With this approach, fragments are complemented by loose information on the associations existing among groups of tuples in them. Loose associations can allow the provider to loosely join the fragments that need to be accessed for evaluating a query. Clearly, the larger the cardinality of the defined groups, the higher the protection, and the less precise query evaluation can be.

4.2 Selective access

Whenever the owner of a dataset outsourced to the cloud needs to selectively share it, it is necessary to enforce the authorization policy set by the owner, to ensure that each user can access all and only the data for which she is authorized. In the following, we discuss the problem of enforcing selective access to outsourced data, and possible approaches for it.

Problem. Whenever the data owner needs to selectively grant access to her data stored on the cloud, there is the need to ensure that her authorization policy be properly enforced. The problem is that neither the data owner (for performance/overhead reasons) nor the cloud provider (for security/trust reasons) can mediate access requests to grant or deny access.

Solutions. Owner-side encryption traditionally assumes that the whole data collection is encrypted using a single secret key, shared between the data owner and authorized users. While protecting data confidentiality, such an approach enables every authorized user to decrypt the entire data collection, hence violating authorizations. A simple, while effective, solution for enforcing selective access to sensitive data stored in the cloud relies on *selective encryption*, meaning that different data items are encrypted with different encryption keys.[19,48] Encryption keys are then distributed to users in such a way that each user can decrypt all and only the data she is authorized to access. To mitigate the burden of key management, selective encryption can be complemented by key derivation,[49] which permits to derive the value of an encryption key starting from the knowledge of another key and of a public piece of information. Key derivation allows each user to manage a single key, from which she can compute all the keys for the resources for which she is authorized. The adoption of selective encryption, while effective for access control enforcement, raises new issues when the policy needs to be updated (i.e., when users are granted/revoked access to data items). Indeed, every policy update naturally translates into a re-encryption of the data item with a key known to the new set of users authorized for it. Since re-encryption should be performed at the data owner side, policy updates can cause considerable overhead. To partially delegate policy updates to the cloud provider, preventing owner-side re-encryption, data can be wrapped in two encryption layers: one enforcing the initial policy defined by the data owner, and a second one enforcing policy updates and enforced by the cloud provider.[48] Updates to the policy can then be delegated to the provider that can re-wrap the owner-encrypted data as needed. Selective encryption and key derivation can be profitably used also to enforce write authorizations[50] and subscription-based policies.[51]
Other approaches aimed at enforcing access control restriction use *attribute-based encryption* (ABE[52]). ABE is a public-key cryptographic schema able to enforce authorization policies defined on *attributes* associated with users,[53] representing characteristics of interest for the authorizations. To this aim, data are associated with *access structures* representing authorization policies, that is, trees where leaves model conditions over attributes, and internal nodes represent logic gates among them. Authorization enforcement is obtained through ABE key generation technique, where the key of a user can decrypt a data item if only if the user's attributes satisfy the access structure of the same. Note that there are also ABE techniques that associate attributes with data and access structures with users.[52]

5. QUERY CONFIDENTIALITY AND INTEGRITY

Besides protecting data, when moving resources and computations to the cloud, it is necessary to protect queries (i.e., accesses and, more in general, computations) operating on such data. In particular, it is necessary to

provide solutions for both protecting query confidentiality (Section 5.1) and assessing the integrity of query results (Section 5.2).

5.1 Query confidentiality

Revealing to the cloud provider, or to an external observer, the target of an access operation (e.g., of a query) may reveal sensitive information about the user formulating the request and/or about the data content, which may be considered confidential. In the following, we discuss the problem and possible approaches for protecting the confidentiality of accesses (and patterns thereof).

Problem. Delegating query evaluation and access operations to an external cloud provider raises security and privacy issues that need to be properly addressed. For instance, the specific target of a data access can reveal (sensitive) information about both the user formulating the query and the accessed value.[54–56] Consider, as an example, a medical dataset stored and managed by a cloud provider. Even if the dataset is public, revealing to an observer (e.g., to the provider) the fact that *Alice* searched new treatments for breast cancer clearly reveals the fact that she is interested in this, and she (or a person close to her) probably suffers from such a disease. Also in case the outsourced dataset is confidential and hence encrypted at the owner side, revealing the target of access requests could also reveal to an external observer the underlying content of the dataset. Assume, as an example, that the observer knows that every night at midnight a transaction reads the balance of a bank account. By monitoring access operations to the dataset, the observer can easily identify the data item corresponding to the target account. As another example, if the frequency of accesses to data is publicly known, the cloud provider can easily match such frequency distribution with the one it observes on encrypted data and infer the content of encrypted resources based on access frequencies. To adequately protect both users and data privacy, it is then necessary to protect also the confidentiality of accesses (i.e., of the target of each access operation) and of patterns thereof (i.e., of the fact that two accesses have the same/a different target).

Solutions. The problem of protecting access and pattern confidentiality has been first addressed by solutions based on *Private Information Retrieval* (PIR).[57] PIR-based approaches focus on protecting access confidentiality, while not protecting the confidentiality of the outsourced dataset, which is assumed to be publicly available. These approaches traditionally suffer from high computational costs, which limit their applicability in real-world scenarios. A second line of works for providing access and pattern confidentiality is based on the adoption of dynamically-allocated data structures. A dynamically allocated data structure is a data structure that changes the allocation of data to memory blocks at each access operation, in such a way that two subsequent accesses targeting the same data read different physical memory blocks. Among dynamically allocated data structure, we can distinguish between *ORAM*-based and *tree*-based solutions. An ORAM (Oblivious-RAM[58–60]) is a structure organized in layers, with a pyramid-shape layout. Each layer l is composed of 2^l buckets, each storing both real and dummy data. Each layer is associated with a function that returns, for each key value, the bucket in the layer where the corresponding entry can be found (if stored in the layer). Every search operation visits the whole data structure, layer by layer, downloading the bucket where the value of interest is possibly stored. The search always reaches the bottom layer (even if the value is found in a high layer of the structure), not to disclose the bucket where the target data has been found. Each accessed bucket is re-written back at the server and the target data item is inserted in one of the two buckets at the first layer. When those buckets are full, the first two layers of the ORAM structure are reconstructed (redefining the corresponding search functions), moving all the data items to the second level. This process is repeated recursively every time a layer in the structure becomes full. Since this approach causes a considerable overhead when the bottom layer needs to be reconstructed, recent ORAM-based approaches (e.g., Path-ORAM[61] and Ring-ORAM[62]) enhanced the underlying data structure and the search process to improve access efficiency. An alternative solution to ORAM-based structures is represented by the *shuffle index*,[63,64] which is a dynamically allocated tree structure. Intuitively, data are stored in the leaves of an unchained B+-tree, which is stored in encrypted form at the cloud provider. To hide the target of access requests, the shuffle index combines three protection techniques: cover searches, cache, and shuffling. Cover searches are fake searches, not recognizable as such, performed in parallel to the search for the target of the access operation. The cache is a local storage structure that temporarily stores, at the client side, the target paths of the most recent accesses. Shuffling consists in dynamically changing the node-block allocation of accessed nodes, be them along the path to the target, to a cover, or in cache. The combined adoption of these

techniques guarantees that different accesses with the same target access different blocks and vice-versa. The shuffle index approach has been extended to leverage the presence of multiple servers.[65] A further extension includes support for selective access.[66]

5.2 Query integrity

Besides guaranteeing that the target of access requests is not revealed to the cloud provider (as well as to external observers), it is necessary to have solutions for providing integrity guarantees to query results. In the following, we discuss this problem and possible approaches for it.

Problem. Whenever query evaluation (or, more in general, computations) is delegated to an external cloud provider, the requesting final user needs techniques for assessing the integrity of the obtained result. Indeed, a lazy provider could operate on a subset (or on an old version) of the data, or even perform a simplified computation to limit its costs. The user formulating the query then needs simple and effective approaches enabling her to verify that the query result be correct (i.e., the query has been properly evaluated and the result is the expected one), complete (i.e., the query has been evaluated on the complete dataset), and fresh (i.e., the query has been evaluated on the latest version of the data).

Solutions. The solutions proposed for verifying the correctness and completeness of query results can be classified in two main categories: *deterministic* techniques and *probabilistic* techniques. Deterministic approaches[28,67,68] are based on the definition of authenticated data structures (e.g., signature chains, skip lists, Merkle hash trees) that can be used to verify whether the result of a query is complete with full confidence in the verification result. With reference to relational datasets, authenticated data structures are defined over one (or a subset) of the attributes of the outsourced relation and can be used to verify the integrity of the results only of queries formulated over these attributes. Therefore, even if they provide fully reliable verification results, these techniques are not flexible enough to support integrity verification for arbitrary queries. Probabilistic approaches, on the contrary, aim at providing flexible solutions that can be suited for arbitrary queries, at the price of reducing the confidence in the verification result.[36,37,69–71] Probabilistic solutions are typically based on the injection, in the original relation, of additional fake tuples (e.g., *markers*[36]) and on the duplication of a set thereof (e.g., *twins*[36]). Since the user knows exactly which tuples have been inserted and/or duplicated, she can verify the presence (or contribution) of such tuples in the query result. The absence of the expected control tuples from the result computed by the cloud provider signals the incompleteness of the result itself. Clearly, fake tuples and duplicates should not be recognizable as such by the cloud provider, which could otherwise elaborate only verification tuples while discarding all the others without being detected. The injection of verification tuples in the data collection clearly implies an additional query evaluation cost, in terms of computational overhead at the cloud provider (which should elaborate verification tuples) and communication overhead of query results (which become larger due to the presence of verification tuples). On the other hand, the injection of a higher number of fake or duplicated tuples naturally increases the probability for the user to detect possible omissions by the cloud provider. The data owner then needs to properly balance the number of verification tuples to find a good trade-off between the overhead they cause and the advantage in terms of the ability to detect incomplete results.

6. COLLABORATIVE QUERY EXECUTION

In scenarios where data processing is delegated to the cloud, it can prove beneficial to distribute the computation among different collaborative providers. In this section, we discuss the problem and possible approaches for collaborative query execution.

Problem. Several scenarios are today characterized by the need of performing computations that involve and combine different informations sources. Whenever such computations are heavy, it can be beneficial to delegate them to cloud providers to leverage their elastic and powerful computational capabilities. If different cloud providers can be collaboratively involved in the computation, the requesting subject can have further economic savings: for example, non-fully trusted but affordable computational providers can be involved in parts of the computation. However, data collections might include sensitive data, and hence access to those data should obey

the authorizations set by the respective data owners. A further problem arising in collaborative computations is that they can entail indirect information flows, which should be properly regulated. For instance, the result of a certain operation over a dataset can disclose information on the original dataset itself, and hence a subject could be authorized for the result only if it is authorized also for the original dataset. For example, in the context of relational databases, consider a query of the form 'SELECT A FROM R WHERE B='10'': while containing only A in its schema, the knowledge of the query result, together with the selection condition inevitably leaks information on the values of attribute B (i.e., 10) as well. Besides against the direct information flows, authorizations should also be enforced against the indirect ones.

Solutions. Access restrictions can be enforced in different manners, including views,[25,72,73] access patterns,[74] and data masking.[75] These approaches, however, only consider explicit information exchanges. To capture both the direct (i.e., explicitly visible) and indirect (i.e., caused by the execution of an operation and possibly not explicitly visible) information flows that a computation can entail, a novel authorization model has been designed[76] that restricts access and shares distributed data considering the implicit information flows and supporting the explicit consideration of join paths in the authorizations. Join paths indicate the join conditions evaluated in the computation of a relation that, while reducing the number of tuples in the relation, cause an indirect information flows (i.e., the fact that the tuple appears in both the relations in the join operation) that should be explicitly authorized. Since authorizations can be formulated by different data authorities, controlling different portions of the data, a solution enabling the composition of authorizations and the verification of information flows against (composed) authorization has been proposed.[77] Also, since different join evaluation strategies imply different information flows among the involved parties, a related proposal considers the adopted join evaluation strategy in the enforcement of access restrictions.[78]
The classical two visibility levels (i.e., no visibility and full visibility) over a data item of authorization models have been extended with a third one (i.e., encrypted visibility), which authorizes a subject to access a piece of data only in encrypted form.[79] Adopting encryption schemes that permit the execution of a certain operation (say, a count required by a query), then the operation can be delegated to non-fully trusted (and possibly affordable) subjects, with clear economic savings. The authorization model is then complemented by a mechanism for the evaluation of the information flow enacted by the execution of the operations involved in relational queries, and for the assignment of data and operations to subjects in such a way that access restrictions be respected with respect to both the explicit as well as the implicit information flows. Another line of works have addressed the problem of specifying privacy requirements (e.g., on how a portion of a query should be executed, for example retrieving data from a certain replica rather than another) that should be satisfied in the definition of a distributed query evaluation strategy.[80,81]

7. CONCLUSIONS

In this paper, we illustrated the main issues related to data protection arising when moving data and computations to an external cloud provider. For each of the discussed problems, we also descibed some of the main existing solutions.

ACKNOWLEDGMENTS

This work was supported in part by the EC within the H2020 Program under grant agreement 825333 (MOSAICrOWN).

REFERENCES

[1] Jhawar, R., Piuri, V., and Santambrogio, M., "Fault tolerance management in cloud computing: a system-level perspective," *IEEE Systems Journal (ISJ)* **7**, 288–297 (June 2013).

[2] Samarati, P. and De Capitani di Vimercati, S., "Cloud security: Issues and concerns," in [*Encyclopedia on Cloud Computing*], Murugesan, S. and Bojanova, I., eds., Wiley (2016).

[3] De Capitani di Vimercati, S., Foresti, S., Livraga, G., Piuri, V., and Samarati, P., "Supporting user requirements and preferences in cloud plan selection," *IEEE Transactions on Services Computing (TSC)* (2017).

[4] Garg, S. K., Versteeg, S., and Buyya, R., "A framework for ranking of cloud computing services," *Future Generation Computer Systems* **29**(4), 1012–1023 (2013).

[5] Li, J., Squicciarini, A. C., Lin, D., Sundareswaran, S., and Jia, C., "MMB^{cloud}-tree: Authenticated index for verifiable cloud service selection," *IEEE Transactions on Dependable and Secure Computing (TDSC)* **14**(2), 185–198 (2017).

[6] Li, A., Yang, X., Kandula, S., and Zhang, M., "CloudCmp: Comparing public cloud providers," in [*Proc. of ACM IMC 2010*], (November 2010).

[7] Luna, J., Suri, N., Iorga, M., and Karmel, A., "Leveraging the potential of cloud security service-level agreements through standards," *IEEE Cloud Computing* **2**(3), 32–40 (2015).

[8] Cloud Security Alliance, "Cloud Control Matrix v3.0.1." https://cloudsecurityalliance.org/working-groups/cloud-controls-matrix. online, accessed on March 19, 2019.

[9] Casola, V., De Benedictis, A., Eraşcu, M., Modic, J., and Rak, M., "Automatically enforcing security SLAs in the cloud," *IEEE Transactions on Services Computing (TSC)* **10**(5), 741–755 (2017).

[10] De Capitani di Vimercati, S., Foresti, S., Livraga, G., and Samarati, P., "Supporting users in data outsourcing and protection in the cloud," in [*International Conference on Cloud Computing and Services Science*], Helfert, M., Ferguson, D., Munoz, V., and Cardoso, J., eds., Springer (2017).

[11] De Capitani di Vimercati, S., Foresti, S., Livraga, G., Piuri, V., and Samarati, P., "A fuzzy-based brokering service for cloud plan selection," *IEEE Systems Journal (ISJ)* (January 2019). to appear.

[12] Foresti, S., Piuri, V., and Soares, G., "On the use of fuzzy logic in dependable cloud management," in [*Proc. of IEEE CNS 2015*], (September 2015).

[13] Arman, A., Foresti, S., Livraga, G., and Samarati, P., "Cloud plan selection under requirements of multiple applications," *Security and Privacy* **1**, e35 (July/August 2018).

[14] Qu, L., Wang, Y., Orgun, M. A., Liu, L., Liu, H., and Bouguettaya, A., "CCCloud: Context-aware and credible cloud service selection based on subjective assessment and objective assessment," *IEEE Transactions on Services Computing (TSC)* **8**(3), 369–383 (2015).

[15] Zheng, Z., Wu, X., Zhang, Y., Lyu, M. R., and Wang, J., "QoS ranking prediction for cloud services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **24**(6), 1213–1222 (2013).

[16] De Capitani di Vimercati, S., Livraga, G., Piuri, V., Samarati, P., and Soares, G., "Supporting application requirements in cloud-based IoT information processing," in [*Proc. of IoTBD 2016*], (April 2016).

[17] De Capitani di Vimercati, S., Livraga, G., and Piuri, V., "Application requirements with preferences in cloud-based information processing," in [*Proc. of IEEE RTSI 2016*], (September 2016).

[18] Jhawar, R., Piuri, V., and Samarati, P., "Supporting security requirements for resource management in cloud computing," in [*Proc. of CSE 2012*], (December 2012).

[19] De Capitani di Vimercati, S., Foresti, S., Livraga, G., and Samarati, P., "Practical techniques building on encryption for protecting and managing data in the cloud," in [*The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*], Ryan, P., Naccache, D., and Quisquater, J.-J., eds., Springer (2016).

[20] Hacigümüs, H., Iyer, B., Mehrotra, S., and Li, C., "Executing SQL over encrypted data in the database-service-provider model.," in [*Proc. of SIGMOD 2002*], (June 2002).

[21] Popa, R., Redfield, C., Zeldovich, N., and Balakrishnan, H., "CryptDB: Protecting confidentiality with encrypted query processing," in [*Proc. of SOSP 2011*], (October 2011).

[22] De Capitani di Vimercati, S., Foresti, S., Livraga, G., Paraboschi, S., and Samarati, P., "Confidentiality protection in large databases," in [*A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*], Flesca, S., Greco, S., Masciari, E., and Saccà, D., eds., Springer (2017).

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

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

[25] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P., "Fragmentation in presence of data dependencies," *IEEE Transactions on Dependable and Secure Computing (TDSC)* **11**(6), 510–523 (2014).

[26] Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "Keep a few: Outsourcing data while maintaining confidentiality," in [*Proc. of ESORICS 2009*], (September 2009).

[27] Hacigümüs, H., Iyer, B., and Mehrotra, S., "Ensuring integrity of encrypted databases in database as a service model," in [*Proc. of DBSec 2003*], (August 2003).

[28] Mykletun, E., Narasimha, M., and Tsudik, G., "Authentication and integrity in outsourced databases," *ACM Transactions on Storage (TOS)* **2**, 107–138 (May 2006).

[29] Boneh, D., Gentry, C., Lynn, B., and Shacham, H., "Aggregate and verifiably encrypted signatures from bilinear maps," in [*Proc. of EUROCRYPT 2003*], (May 2003).

[30] Mykletun, E., Narasimha, M., and Tsudik, G., "Signature bouquets: Immutability for aggregated/condensed signatures," in [*Proc. of ESORICS 2004*], (September 2004).

[31] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., and Song, D., "Provable data possession at untrusted stores," in [*Proc. of ACM CCS 2007*], (October/November 2007).

[32] Juels, A. and Kaliski, B., "PORs: Proofs of retrievability for large files," in [*Proc. of ACM CCS 2007*], (October–November 2007).

[33] Ateniese, G., Burns, R., Curtmola, R., Herring., J., Kissner, O. K. L., Peterson, Z., and Song, D., "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC)* **14**, 12:1–12:34 (June 2011).

[34] Shacham, H. and Waters, B., "Compact proofs of retrievability," in [*Proc. of ASIACRYPT*], (December 2008).

[35] Yuan., J. and Yu, S., "Proofs of retrievability with public verifiability and constant communication cost in cloud," in [*Proc. of International Workshop on Security in Cloud Computing*], (May 2013).

[36] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "Integrity for join queries in the cloud," *IEEE Transactions on Cloud Computing (TCC)* **1**, 187–200 (July-Dec. 2013).

[37] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P., "Integrity for distributed queries," in [*Proc. of IEEE CNS 2014*], (October 2014).

[38] Wang, C., Chow, S., Wang, Q., Ren, K., and Lou, W., "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers (TC)* **62**, 362–375 (February 2013).

[39] Basescu, C., Cachin, C., Eyal, I., Haas, R., Sorniotti, A., Vukolic, M., and Zachevsky, I., "Robust data sharing with key-value stores," in [*Proc. of DNS 2012*], (June 2012).

[40] Bessani, A., Correia, M., Quaresma, B., Andre, F., and Sousa, P., "Depsky: Dependable and secure storage in a cloud-of-clouds," in [*Proc. of EuroSys 2011*], (April 2011).

[41] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "On information leakage by indexes over data fragments," in [*Proc. of PrivDB 2013*], (April 2013).

[42] Wang, H. and Lakshmanan, L., "Efficient secure query evaluation over encrypted XML databases," in [*Proc. of VLDB 2006*], (September 2006).

[43] Wang, C., Cao, N., Ren, K., and Lou, W., "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems (TPDS)* **23**, 1467–1479 (August 2012).

[44] Gentry, C., "Fully homomorphic encryption using ideal lattices," in [*Proc. of STOC 2009*], (May 2009).

[45] Popa, R., Redfield, C., Zeldovich, N., and Balakrishnan, H., "CryptDB: Protecting confidentiality with encrypted query processing," in [*Proc. of SOSP 2011*], (October 2011).

[46] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "Fragments and loose associations: Respecting privacy in data publishing," *Proc. of VLDB Endowment (PVLDB)* **3**, 1370–1381 (September 2010).

[47] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P., "Loose associations to increase utility in data publishing," *Journal of Computer Security (JCS)* **23**(1), 59–88 (2015).

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

[49] Atallah, M., Blanton, M., Fazio, N., and Frikken, K., "Dynamic and efficient key management for access hierarchies," *ACM Transactions on Information and System Security (TISSEC)* **12**, 18:1–18:43 (January 2009).

[50] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P., "Enforcing dynamic write privileges in data outsourcing," *Computers & Security* **39**, 47–63 (November 2013).

[51] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., and Livraga, G., "Enforcing subscription-based authorization policies in cloud scenarios," in [*Proc. of DBSec 2012*], (July 2012).

[52] Goyal, V., Pandey, O., Sahai, A., and Waters, B., "Attribute-based encryption for fine-grained access control of encrypted data," in [*Proc. of ACM CCS 2006*], (October/November 2006).

[53] Waters, B., "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in [*Proc. of PKC 2011*], (March 2011).

[54] Islam, M. S., Kuzu, M., and Kantarcioglu, M., "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in [*Proc. of NDSS 2012*], (February 2012).

[55] Kellaris, G., Kollios, G., Nissim, K., and O'Neill, A., "Generic attacks on secure outsourced databases," in [*Proc. of ACM CCS 2016*], (Oct. 2016).

[56] Naveed, M., Kamara, S., and Wright, C. V., "Inference attacks on property-preserving encrypted databases," in [*Proc. of ACM CCS 2015*], (Oct. 2015).

[57] Ostrovsky, R. and Skeith, W. E., "A survey of single-database private information retrieval: Techniques and applications," in [*Proc. of PKC 2007*], (Apr. 2007).

[58] Goldreich, O., "Towards a theory of software protection and simulation by Oblivious RAMs," in [*Proc. of STOC 1987*], (May 1987).

[59] Ostrovsky, R., "Efficient computation on Oblivious RAMs," in [*Proc. of STOC 1990*], (May 1990).

[60] Goldreich, O. and Ostrovsky, R., "Software protection and simulation on Oblivious RAMs," *Journal of the ACM* **43**, 431–473 (May 1996).

[61] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C. W., Ren, L., Yu, X., and Devadas, S., "Path ORAM: an extremely simple Oblivious RAM protocol," in [*Proc. of ACM CCS 2013*], (Nov. 2013).

[62] Ren, L., Fletcher, C. W., Kwon, A., Stefanov, E., Shi, E., van Dijk, M., and Devadas, S., "Constants count: Practical improvements to Oblivious RAM," in [*Proc. of USENIX 2015*], (Aug. 2015).

[63] De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., and Samarati, P., "Efficient and private access to outsourced data," in [*Proc. of ICDCS 2011*], (June 2011).

[64] De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., and Samarati, P., "Shuffle index: Efficient and private access to outsourced data," *ACM Transactions on Storage (TOS)* **11**, 1–55 (October 2015). Article 19.

[65] De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., and Samarati, P., "Three-server swapping for access confidentiality," *IEEE Transactions on Cloud Computing (TCC)* **6**, 492–505 (April-June 2018).

[66] De Capitani di Vimercati, S., S.Foresti, Paraboschi, S., Pelosi, G., and Samarati, P., "Enforcing authorizations while protecting access confidentiality," *Journal of Computer Security (JCS)* **26**, 143–175 (January 2018).

[67] Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L., "Dynamic authenticated index structures for outsourced databases," in [*Proc. of SIGMOD 2006*], (Jun. 2006).

[68] Pang, H. and Tan, K., "Verifying completeness of relational query answers from online servers," *ACM Transactions on Information and System Security (TISSEC)* **11**, 5:1–5:50 (May 2008).

[69] Ghazizadeh, P., Mukkamala, R., and Olariu, S., "Data integrity evaluation in cloud database-as-a-service," in [*Proc. of IEEE SERVICES*], 280–285 (June 2013).

[70] Wang, H., Yin, J., Perng, C., and Yu, P., "Dual encryption for query integrity assurance," in [*Proc. of CIKM 2008*], (Oct. 2008).

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

[72] Guarnieri, M. and Basin, D., "Optimal security-aware query processing," *Proc. of VLDB Endowment (PVLDB)* **7**(12), 1307–1318 (2014).

[73] Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P., "Extending query rewriting techniques for fine-grained access control," in [*Proc. of SIGMOD*], 551–562 (June 2004).

[74] Benedikt, M., Leblay, J., and Tsamoura, E., "Querying with access patterns and integrity constraints," *Proc. of VLDB Endowment (PVLDB)* **8**(6), 690–701 (2015).

[75] Kwakye, M. M. and Barker, K., "Privacy-preservation in the integration and querying of multidimensional data models," in [*Proc of PST 2016*], 255–263 (December 2016).

[76] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "Authorization enforcement in distributed query evaluation," *Journal of Computer Security (JCS)* **19**(4), 751–794 (2011).

[77] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P., "Assessing query privileges via safe and efficient permission composition," in [*Proc. of ACM CCS 2008*], (October 2008).

[78] Zeng, Q., Zhao, M., Liu, P., Yadav, P., Calo, S., and Lobo, J., "Enforcement of autonomous authorizations in collaborative distributed query evaluation," *IEEE Transactions on Knowledge and Data Engineering (TKDE)* **27**(4), 979–992 (2015).

[79] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., and Samarati, P., "An authorization model for multi-provider queries," *Proc. of the VLDB Endowment* **11** (November 2017).

[80] Farnan, N., Lee, A., Chrysanthis, P., and Yu, T., "Don't reveal my intension: Protecting user privacy using declarative preferences during distributed query processing," in [*Proc. of ESORICS 2011*], (September 2011).

[81] Farnan, N., Lee, A., Chrysanthis, P., and Yu, T., "PAQO: Preference-aware query optimization for decentralized database systems," in [*Proc. of ICDE 2014*], (March–April 2014).