Data Security and Privacy in the Cloud

Pierangela Samarati

Università degli Studi di Milano
Dipartimento di Informatica
Via Bramante 65 – 26013 Crema – Italy
pierangela.samarati@unimi.it

Abstract. Achieving data security and privacy in the cloud means ensuring confidentiality and integrity of data and computations, and protection from non authorized accesses. Satisfaction of such requirements entails non trivial challenges, as relying on external servers, owners lose control on their data. In this paper, we discuss the problems of guaranteeing proper data security and privacy in the cloud, and illustrate possible solutions for them.

Keywords: Cloud computing, confidentiality, integrity, honest-but-curious servers, data fragmentation, inferences, private access, shuffle index, query integrity.

1 Introduction

Cloud computing has emerged as a successful paradigm increasingly appealing to individuals and companies for storing, accessing, processing, and sharing information. The cloud provides, in fact, significant benefits of scalability and elasticity, allowing its users to conveniently offer and enjoy services at reduced costs thanks to the economy of scale that providers can exploit. Relying on the cloud for storing and managing data brings, together with all the benefits and convenience, also new security and privacy risks (e.g., [20,25,27,34]). In this paper, we address in particular the problems related to the protection of data and of computations on them. On one hand, cloud providers can be assumed to employ basic security mechanisms for protecting data outsourced to them, maybe even employing controls that would not be affordable by most individuals or small companies. On the other hand, however, relying on external parties for storing or processing data, users lose control on such data hence leaving them potentially exposed to security and privacy risks. Data could be sensitive and should be maintained confidential even with respect to the cloud provider itself that, while trustworthy for providing services, should not be allowed to know the actual data content (*honest-but-curious* servers). Even the integrity of data – or of computations on them – can be at risk as providers might behave not correctly (*lazy* or *malicious* servers).

X. Huang and J. Zhou (Eds.): ISPEC 2014, LNCS 8434, pp. 28–41, 2014.

Protecting data and computations entail then ensuring both confidentiality and integrity. Confidentiality issues arise since data externally stored or managed can contain sensitive information, or information that the owner wishes to maintain confidential. Confidentiality should be guaranteed also with respect to the server storing or processing data, hence introducing complications in query execution and in the enforcement of possible access restrictions. In addition to the stored data, even users' accesses on them may need to be maintained confidential. Integrity issues arise since external servers storing or processing data could misbehave. Guaranteeing integrity requires providing users with the ability of assessing that data are stored correctly, computations are performed correctly, and returned results are correct.

This paper is organized in two main sections. Section 2 illustrates confidentiality issues, presenting solutions for guaranteeing confidentiality of data in storage, enforcing access restrictions, and guaranteeing confidentiality of accesses over data. Section 3 illustrates integrity issues, presenting solutions for guaranteeing integrity of data as well as of computations on them.

2 Confidentiality of Data and Access Control

Guaranteeing confidentiality in the cloud entails ensuring confidentiality to the stored data (Section 2.1), enforcing of access restrictions on the data (Section 2.2), and maintaining confidentiality of the accesses performed on the data (Section 2.3).

2.1 Encryption and Fragmentation

A natural solution for protecting data confidentiality consists in *encrypting* data before releasing them to the external server for storage. Encryption can be applied at different granularity levels. In particular, when data are organized in relational tables, encryption can be applied at the level of *table*, *attribute*, *tuple*, and *cell* [7,22]. Encryption at the level of tuple appears to be preferred as it provides some support for fine-grained access while not requiring too many encryption/decryption operations. Also, for performance reasons, symmetric encryption is usually adopted.

A complication in dealing with encrypted data is that, since the server storing the data should not know the actual data content, data cannot be decrypted for query execution. A possible solution to this obstacle consists in evaluating queries on the encrypted values themselves. For instance, homomorphic encryption solutions provide such a capability but support limited kinds of queries and suffer from high performance overhead. An alternative solution consists in associating with the encrypted tuples some metadata, called *indexes*, which can be used for query execution (e.g., [7,22]). Intuitively an index column can be specified for every attribute on which conditions need to be evaluated. Conditions on plaintext values, known at the trusted client side, are then translated

PATIENT

SSN	Name	YoB	Job	Disease	Treatment
123-45-6789	A. Allen	1971	hairdressing	eczema	ointments
635-98-3692	B. Brown	1954	painter	asthma	bronchodilator
820-73-0735	C. Clark	1985	plastic worker	dermatite	corticosteroids
838-91-9634	D. Davis	1962	miners	silicosis	oxygen
168-87-4067	E. Evans	1977	lab techn	hepatitis	antiviral drug
912-83-7265	F. Fisher	1960	nurse	tuberculosis	antibiotics

(a)

PATIENT^e

tid	enc	$\mathbf{I}_n$	$\mathbf{I}_y$	$\mathbf{I}_j$	$\mathbf{I}_d$
1	τ	n_1	y_1	j_1	d_1
2	σ	n_2	y_1	j_2	d_2
3	λ	n_3	y_1	j_2	d_1
4	ρ	n_4	y_2	j_3	d_2
5	α	n_5	y_3	j_1	d_2
6	δ	n_6	y_3	j_3	d_2

(b)

Fig. 1. An example of plaintext relation (a) and of corresponding encrypted and indexed relation (b)

into conditions on index values to be evaluated at the server side. Figure 1 illustrates an example of relation and its corresponding encrypted and indexed version, where indexes have been defined over attributes `Name`, `YoB`, `Job`, and `Disease` of the plaintext relation. Query execution is enforced by: a query to be executed by the server over the index values and a further query to be executed by the client on the server's result once decrypted, to evaluate further conditions and producing the actual result. Index values should be well related to the plaintext values to provide effective in query execution, while at the same time not leak information on the plaintext values behind them. Different indexing functions have been proposed that differ in how plaintext values are mapped onto index values, such as [15]: *direct* indexing, providing a one-to-one mapping between plaintext values and index values; *hash and bucket-based* indexing, providing a many-to-one mapping between plaintext values and index values (thus generating collisions); and *flattened* indexing, providing a one-to-many mapping between plaintext values and index values (to not expose the indexing function to frequency-based inferences). All these types of indexing functions differ in the offered protection guarantees, the kinds of queries supported, and the performance overhead suffered. For instance, direct indexing allows precise evaluation of equality conditions, and even of range conditions if indexes are ordered but is also the one most exposed to inference attacks compromising the confidentiality of the indexing function.

As encryption makes query evaluation more complex or not always possible, alternative solutions have been devised trying to limit encryption, or depart from it. In particular, when what is sensitive are not the data values themselves but their association, confidentiality can be guaranteed breaking the association (i.e., its visibility) by storing the involved attributes in separate *data fragments*. The association is then protected by restricting visibility of the fragments or ensuring their un-linkability. A sensitive association can be represented as a set of attributes whose joint visibility (i.e., whose association) is sensitive. Attributes whose values are sensitive are also captured in such representation as they correspond to singleton sets. Figure 2(a) illustrates an example of relation and confidentiality constraints over it. Different fragmentation paradigms have been proposed differing in the use of encryption and in the assumptions required to ensure protection of the sensitive associations.

PATIENT

SSN	Name	YoB	Job	Disease	Treatment
123-45-6789	A. Allen	1971	hairdressing	eczema	ointments
635-98-3692	B. Brown	1954	painter	asthma	bronchodilator
820-73-0735	C. Clark	1985	plastic worker	dermatite	corticosteroids
838-91-9634	D. Davis	1962	miners	silicosis	oxygen
168-87-4067	E. Evans	1977	lab techn	hepatitis	antiviral drug
912-83-7265	F. Fisher	1960	nurse	tuberculosis	antibiotics

$c_1 = \{\texttt{SSN}\}$
$c_2 = \{\texttt{Name},\texttt{Disease}\}$
$c_3 = \{\texttt{Name},\texttt{Job}\}$
$c_4 = \{\texttt{Job},\texttt{Disease}\}$

(a) Original relation and confidentiality constraints

F_1

tid	Name	YoB	Treatment	SSN^{1e}	Disease^{1e}
1	A. Allen	1971	ointments	α	η
2	B. Brown	1954	bronchodilator	β	ρ
3	C. Clark	1985	corticosteroids	γ	σ
4	D. Davis	1962	oxygen	δ	π
5	E. Evans	1977	antiviral drugs	ϵ	ϕ
6	F. Fisher	1960	antibiotics	θ	ε

F_2

tid	Job	SSN^{2e}	Disease^{2e}
1	hairdressing	χ	ξ
2	painter	τ	η
3	plastic worker	η	ζ
4	miners	ν	λ
5	lab techn	μ	ϱ
6	nurse	ω	ι

(b) Two can keep a secret

F_1

salt	enc	Name	YoB
s_{11}	Δ	A. Allen	1971
s_{12}	ν	B. Brown	1954
s_{13}	ρ	C. Clark	1985
s_{14}	σ	D. Davis	1962
s_{15}	ε	E. Evans	1977
s_{16}	π	F. Fisher	1960

F_2

sal	enc	Disease	Treatment
s_{21}	β	eczema	ointments
s_{22}	γ	asthma	bronchodilator
s_{23}	δ	dermatite	corticosteroids
s_{24}	μ	silicosis	oxygen
s_{25}	ϵ	hepatitis	antiviral drug
s_{26}	χ	tuberculosis	antibiotics

F_3

sal	enc	Job
s_{31}	ψ	hairdressing
s_{32}	ω	painter
s_{33}	Σ	plastic worker
s_{34}	Π	miners
s_{35}	λ	lab techn
s_{36}	ι	nurse

(c) Multiple fragments

F_o

tid	SSN	Name	Job
1	123-45-6789	A. Allen	hairdressing
2	635-98-3692	B. Brown	painter
3	820-73-0735	C. Clark	plastic worker
4	838-91-9634	D. Davis	miners
5	168-87-4067	E. Evans	lab techn
6	912-83-7265	F. Fisher	nurse

F_s

tid	YoB	Disease	Treatment
1	1971	eczema	ointments
2	1954	asthma	bronchodilator
3	1985	dermatite	corticosteroids
4	1962	silicosis	oxygen
5	1977	hepatitis	antiviral drug
6	1960	tuberculosis	antibiotics

(d) Keep a few

Fig. 2. A sample relation with confidentiality constraints and its fragmentation with different paradigms

– *Two can keep a secret* [1]. Data are split in two fragments stored at two independent external servers which are assumed to not communicate and not know each other. Sensitive attributes are obfuscated (e.g., encrypted). Sensitive associations are protected by splitting the attributes among the two servers. In addition to sensitive attributes, other attributes may be obfuscated if their plaintext storage at any of the two servers would expose

some sensitive associations. The two fragments have a tuple identifier in common, allowing the data owner to correctly reconstruct the original relation. Figure 2(b) illustrates a sample fragmentation in two fragments for relation PATIENT of Figure 2(a), subject to the reported confidentiality constraints. Note that, while not sensitive by itself, attribute `Disease` is obfuscated since its plaintext storage in any of the two fragments would violate some constraint (c_2 for fragment F_1 and c_4 for fragment F_2).

- *Multiple fragments* [4,6,10,13]. It does not impose any assumption on the external servers or on the number of fragments. Sensitive attributes are stored in encrypted form. Sensitive associations are protected by splitting the involved attributes among different fragments. Fragments are assumed to be complete (every attribute is represented either in plaintext or in the encrypted chunk) and to not have attributes in common (to ensure their unlinkability). The encrypted chunk is produced with salts to avoid exposure of values with multiple occurrences. Figure 2(c) illustrates a sample fragmentation with multiple fragments for relation PATIENT; the use of three fragments permits to represent in plaintext form all attributes that are not sensitive by themselves.
- *Keep a few* [5]. It assumes the data owner (or a trusted party) to store a limited portion of the data and completely departs from encryption (i.e., all attributes are stored in plaintext form). Sensitive attributes are stored at the owner side. Sensitive associations are protected by ensuring that, for each constraint, at least one of the attributes in it is stored at the owner side. A tuple identifier is maintained in both fragments to allow the owner to correctly reconstruct the original relation. Figure 2(d) illustrates a sample fragmentation for relation PATIENT with this approach, where F_o is the fragment stored at the owner side.

The advantage of fragmentation over encryption is the availability of data in the clear and therefore the ability of the server to evaluate any condition on them; by contrast encrypted data or indexes provide limited support for evaluating conditions. In any of the strategies above, fragmentation should be enforced trying to maximize the availability of attributes in the clear and to minimize the fragmentation enforced. Also, additional criteria could be taken into account such as the query workload or possible visibility requirements.

Fragmentation approaches assume fragments to not be linkable (and therefore associations to be protected) when they have no common attributes. In other words, attributes are assumed to be independent. However, often dependencies may exist among attributes introducing inferences from some attributes over others that indirectly expose otherwise not visible attributes or enable linking among fragments [11]. To illustrate, consider the set of attributes and constraints in Figure 3(a), also represented as a graph with one node per attribute and constraint and with multi-arcs connecting attributes to constraints. A fragmentation reporting, `Birth`, `Zip` and `Disease` in one fragment (light gray in the graph) and `Treatment`, `Premium`, and `Insurance` in the other (dark gray in the graph) appears to satisfy the constraints (note that attributes appearing dotted

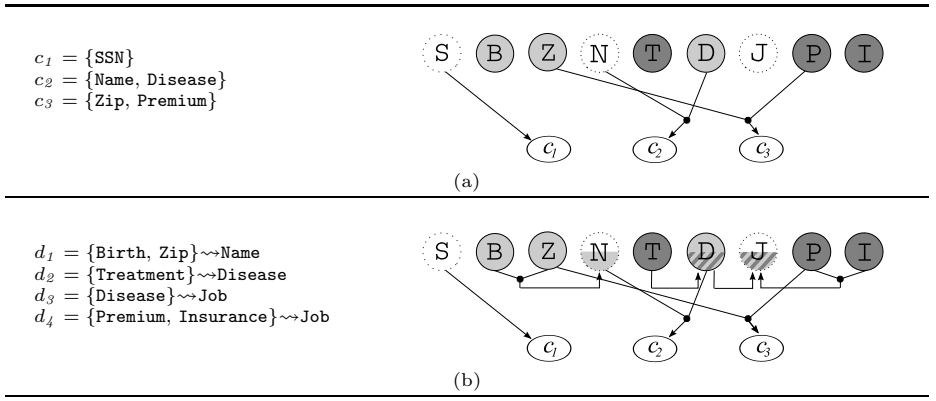

Fig. 3. An example of exposure of sensitive associations due to data dependencies

in the graph are not represented in the clear in any of the fragments). In fact, no sensitive information is exposed in the fragments and not having attributes in common the fragments cannot be linked. This reasoning would be perfectly fine if attributes where independent. However, dependencies might exist and some information be inferable from other, allowing an observer to: from the `Birthdate` and `Zip` reduce uncertainty over the `Name`; from the `Treatment` infer the `Disease`; from the `Disease` reduce uncertainty over the `Job`; and from the `Premium` and the `Insurance` infer the `Job`. Such derivations due to dependencies can indirectly expose information and leak sensitive information or enable linking. Figure 3(b) extends the graph with multi-arcs representing dependencies (from the premises to the consequence) and illustrates such inferences by propagating colors from the premises to the consequence. More precisely, if a given color appears in all the attributes of a premise, it is propagated to the attribute in the consequence (in the graphical representation, propagated colors are reported in the bottom half of the nodes). Multi colored attributes represent (indirect) violations of the constraints.

2.2 Access Control Enforcement

In many scenarios access to data is *selective*, meaning that different users, or groups of them, should have different views/access on the data. With data outsourced to external servers, the problem therefore arises of how to enforce access control. In fact, the data owner cannot mediate every access request, as the advantages of delegating the management of data to an external server would be lost. On the other hand, the server storing the data may not be trusted for the enforcement of the access control policy, which could also be sensitive and should be protected from the server' eyes.

A possible solution to have access control enforced without requiring the data owner intervention at every access consists in combining access control with

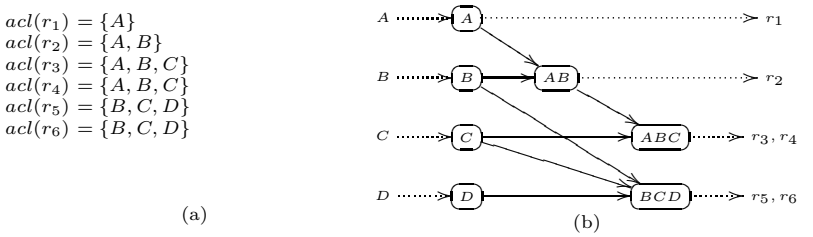

Fig. 4. An example of access control policy (a) and key derivation hierarchy (b) enforcing it

encryption, wrapping the data with a (self-enforcing) protecting layer. Some solutions in this direction rely on attribute-based encryption (ABE), possibly combined with other cryptographic techniques (e.g., [30]). An alternative interesting approach combines access control with encryption by selectively encrypting resources based on the authorizations on them [12]. Intuitively, data are encrypted with different keys and users are given only the keys for data which they are authorized to access. This solution introduces some challenges related to key management: users would like to have a single key (regardless of the number of resources for which they have access), and data should be encrypted at most once (i.e., different replicas with different keys should be avoided). These requirements can be satisfied by adopting a *key derivation* approach [2], by which users can derive keys from a single key assigned to them and public tokens. Access control can then be enforced by properly organizing the keys with which resources are encrypted in a hierarchy reflecting the authorizations on the resources, or better their access control lists (ACLs), where the key corresponding to an ACL allows deriving, via one or more tokens, the keys associated with all ACLs that are superset of it. This way a user is able to derive, from her key and public tokens, all (and only) the keys that are needed to access resources that she is authorized to access according to the access control policy. Figure 4(a) reports an example of access control policy involving four users (A, B, C, and D) and six resources ($r_1, r_2, r_3, r_4, r_5, r_6$). Figure 4(b) reports a corresponding hierarchy for keys, including one key for each user and one key for each non singleton ACL appearing in the policy. Dotted lines connect users to their keys, and keys to resources encrypted with them. Tokens, represented as continuous line, allow users to derive from their own key the keys of all and only resources for which they are authorized. This hierarchy-based approach can also be extended to support write privileges [9], and subscription-based scenarios [8].

As the key with which resources are encrypted depends on their ACL, in principle any change to the authorization policy would require downloading, decrypting, re-encrypting, and re-uploading the involved resources. This process can be avoided assuming some cooperation of the server in enforcing authorization changes by over-encrypting resources to make them not accessible to non-authorized users who know the key with which resources have been encrypted by the owner. Intuitively, every resource is subject to two layers of encryption [12]: a *base encryption layer* (BEL) applied by the data owner reflects the access

control policy at initialization time; a *surface encryption layer* (SEL) applied on top of the BEL by the server takes into account possible changes in the access control policy. A user will be able to access a resource only if she can pass both layers of encryption.

2.3 Private Access

In some scenarios what can be sensitive might be not (or not only) the data stored at the external server but (also) the accesses that users make on such data. In particular, the fact that an access aims at specific data (*access confidentiality*) or that different accesses aim at the same data (*pattern confidentiality*) should be maintained confidential, even to the server providing access itself. Traditional approaches addressing these issues are based on *private information retrieval* (PIR) techniques, which however assume that data are stored in the clear, and suffer from high computation costs, thus limiting their applicability (e.g., [32]). Alternative solutions are based on the Oblivious RAM structure (ORAM) [21] and on dynamic data allocation techniques (e.g., [17,18,35]). These solutions provide access and pattern confidentiality by encrypting data and changing their physical allocation at every access so to destroy the, otherwise static, correspondence between data and physical blocks where they are stored. In particular, *Path ORAM* [35] maintains some data in a local cache (called *stash*) and some data in an external tree structure where nodes contain, in addition to actual blocks, also dummy blocks (to provide uniformity of the size of all nodes). Every access entails reading a path of the tree (containing the searched block) and bringing the nodes in the path in the stash. Then, the nodes in the tree path read are rewritten back (possibly moving out from the stash nodes mapping to leaves intersecting the path).

The *shuffle* index also assumes a local cache but maintains at the external storage the complete data structure (so providing also more resilience in case of failures and accommodating concurrent accesses or distributed scenarios [18,19]). The data stored externally are organized with a B+-tree whose nodes are encrypted and that has no pointer between leaves (to avoid leaking to the server information on the order of values stored). The advantage of a key-based hierarchical organization is that it allows supporting range queries. To protect pattern confidentiality, for every search operation, the client asks retrieval of more values: the actual target and some covers (which provide uncertainty for the server on the block to which the client actually aims). Also, the client performs a *shuffling* among nodes in the cache and those retrieved by the search, then rewriting the involved blocks (whose content have been changed by the re-allocation enforced by the shuffling) on the server. Figure 5(a) illustrates an example of nodes for a shuffle index where, for readability, we have omitted the pointers from a node to its children; the parent-child relationship is however understandable from the label assigned to nodes in the figure, as leaf nodes have as prefix the label of their parent. The figure shows a sample execution with target *c1* and cover *b2*, assuming *a1* (and then the path to it) be in cache. The dotted arrows in Figure 5(a) illustrate a possible shuffling that changes the data allocated to blocks

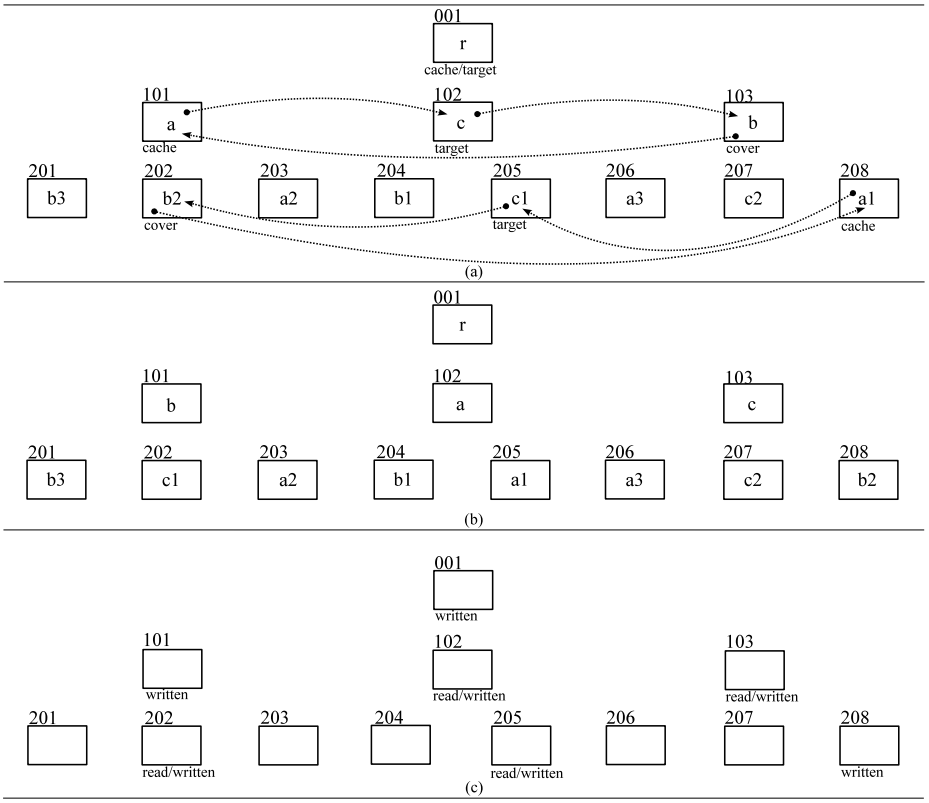

Fig. 5. An example of logical shuffle index with cache/target/cover and shuffling operations due to an access (a), logical shuffle index at the end of the access (b), and server's view on the access (c)

as in Figure 5(b). Figure 5(c) illustrates instead the view of the server on the access.

3 Data and Computation Integrity

Another issue that needs to be considered when storing – or processing – data at external servers is the ability to assess the correct behavior of the servers. This implies verifying that data are maintained correctly and that queries performed on them are correctly executed.

As for data storage, typical solutions are based on hashing and *digital signature schemas* (e.g., [23,31]). Signature-based approaches provide a *deterministic* guarantee of data integrity but impose an overhead not always acceptable in cloud scenarios. Alternative solutions, which provide *probabilistic* (i.e., not certain) guarantees of data integrity, are *Proof of Retrievability* (POR) approaches (e.g., [28]), which apply to encrypted data, or *Provable Data Possession* (PDP)

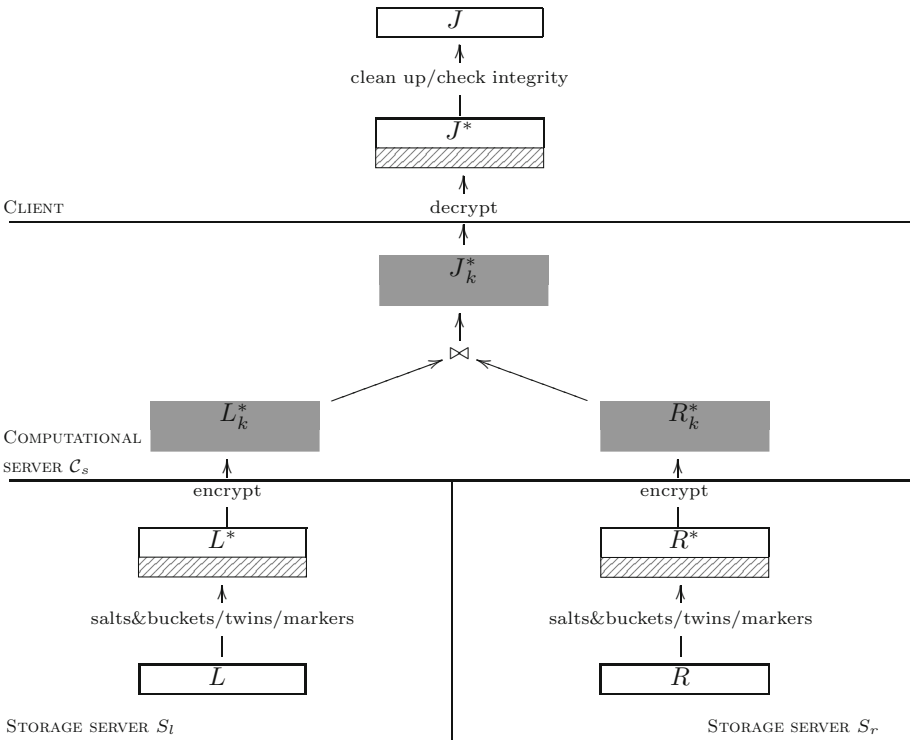

Fig. 6. Join execution and integrity controls

approaches (e.g., [3]), which apply to generic datasets. They are based on control sentinels hidden among the encrypted data (for POR) or on homomorphic verifiable tags (for PDP) whose presence and correctness can be verified by the owner or other trusted parties.

As for query computation, guaranteeing integrity requires to provide users with mechanisms to verify the *correctness*, *completeness*, and *freshness* of computations. Correctness means that the result has been performed on the original data and the computation performed correctly. Completeness means that no data is missing from the result. Freshness means that the computation has been performed on the most recent version of the data. Similarly to data storage solutions, also solutions assessing integrity of computations can be distinguished in *deterministic* and *probabilistic*. Deterministic solutions are based on the use of *authenticated data structures* and allow assessing integrity of query results based on a *verification object* which the server should return together with the results. Different approaches can different with respect to how data are organized and the verification object computed. For instance, signature chaining schemas (e.g., [33]) organize tuples in a chain while Merkle tree and its variation

(e.g., MB-Tree) [29,38] organize data in a hash-based tree structure. Deterministic solutions permit to detect integrity violations with certainty but offer such a capability only for queries with conditions on the attribute/s on which the structure has been defined.

Probabilistic solutions accommodate a more general control, while providing only a probabilistic guarantee of detecting violations (e.g., [14,36,37]). For instance, the proposal in [14] permits to assess the integrity of join queries performed by a non trustworthy computational server. The approach is based on the insertion, in the encrypted data passed to the computational server, of fake tuples (not belonging to the original relations) representing *markers* (newly generated tuples) and *twins* (replicas of existing tuples). To flatten the distribution of tuples participating in a join, tuples can be organized in *buckets* by using *salts* in the encryption, and possibly inserting dummy tuples. A violation is detected if an expected marker is missing or a twinned tuple appears solo. Of course, the more the markers and twins inserted, the more the offered guarantee. While the guarantee is only probabilistic, a confident assurance can be provided with limited performance overhead. Figure 6 illustrates a high level representation of the working of this technique.

4 Conclusions

The use of external cloud services for storing and processing data offers tremendous benefits to companies as well as users. Such a convenience introduces however inevitable risks on the data, and the need to provide techniques for ensuring confidentiality and integrity of data as well as of computations on them. This paper discussed these needs and some solutions addressing them.

Acknowledgments. I would like to thank Sabrina De Capitani di Vimercati for suggestions and comments. This paper is based on joint work with Sabrina De Capitani di Vimercati, Sara Foresti, Giovanni Livraga, Sushil Jajodia, Stefano Paraboschi, and Gerardo Pelosi. The work was supported in part by Italian MIUR PRIN project "GenData 2020" and EC 7FP project ABC4EU (312797).

References

1. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: A distributed architecture for secure database services. In: Proc. of the 2nd Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA (January 2005)
2. Atallah, M., Blanton, M., Fazio, N., Frikken, K.: Dynamic and efficient key management for access hierarchies. ACM Transactions on Information and System Security 12(3), 18:1–18:43 (2009)
3. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proc. of the 14th ACM Conference on Computer and Communications Security (CCS 2007), Alexandria, VA, USA (October-November 2007)

4. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 171–186. Springer, Heidelberg (2007)
5. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a few: Outsourcing data while maintaining confidentiality. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 440–455. Springer, Heidelberg (2009)
6. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC) 13(3), 22:1–22:33 (2010)
7. Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of the 10th ACM Conference on Computer and Communications Security (CCS 2003), Washington, DC, USA (October 2003)
8. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G.: Enforcing subscription-based authorization policies in cloud scenarios. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 314–329. Springer, Heidelberg (2012)
9. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Enforcing dynamic write privileges in data outsourcing. Computers & Security (COSE) 39, 47–63 (2013)
10. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Extending loose associations to multiple fragments. In: Wang, L., Shafiq, B. (eds.) DBSec 2013. LNCS, vol. 7964, pp. 1–16. Springer, Heidelberg (2013)
11. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Fragmentation in presence of data dependencies. IEEE Transactions on Dependable and Secure Computing (TDSC) (2014)
12. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Encryption policies for regulating access to outsourced data. ACM Transactions on Database Systems (TODS) 35(2), 12:1–12:46 (2010)
13. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragments and loose associations: Respecting privacy in data publishing. Proc. of the VLDB Endowment 3(1), 1370–1381 (2010)
14. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Integrity for join queries in the cloud. IEEE Transactions on Cloud Computing (TCC) 1(2), 187–200 (2013)
15. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: On information leakage by indexes over data fragments. In: Proc. of the 1st International Workshop on Privacy-Preserving Data Publication and Analysis (PrivDB 2013), Brisbane, Australia (April 2013)
16. De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P.: Data privacy: Definitions and techniques. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 20(6), 793–817 (2012)
17. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: Proc. of the 31st International Conference on Distributed Computing Systems (ICDCS 2011), Minneapolis, Minnesota, USA (June 2011)

18. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Distributed shuffling for preserving access confidentiality. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 628–645. Springer, Heidelberg (2013)
19. De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Supporting concurrency and multiple indexes in private access to outsourced data. Journal of Computer Security (JCS) 21(3), 425–461 (2013)
20. De Capitani di Vimercati, S., Foresti, S., Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: Proc. of the 7th International Conference on Risks and Security of Internet and Systems (CRiSIS 2012), Cork, Ireland (October 2012)
21. Goldreich, O., Ostrovsky, R.: Software protection and simulation on Oblivious RAMs. Journal of the ACM 43(3), 431–473 (1996)
22. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2002), Madison, Wisconsin, USA (June 2002)
23. Hacigümüş, H., Iyer, B., Mehrotra, S.: Ensuring integrity of encrypted databases in database as a service model. In: De Capitani di Vimercati, S., Ray, I., Ray, I. (eds.) Data and Applications Security XVII. IFIP, vol. 142, pp. 61–74. Springer, Heidelberg (2004)
24. Jhawar, R., Piuri, V.: Adaptive resource management for balancing availability and performance in cloud computing. In: Proc. of the 10th International Conference on Security and Cryptography (SECRYPT 2013), Reykjavik, Iceland (July 2013)
25. Jhawar, R., Piuri, V., Samarati, P.: Supporting security requirements for resource management in cloud computing. In: Proc. of the 15th IEEE International Conference on Computational Science and Engineering (CSE 2012), Paphos, Cyprus (December 2012)
26. Jhawar, R., Piuri, V., Santambrogio, M.: A comprehensive conceptual system-level approach to fault tolerance in cloud computing. In: Proc. of the 2012 IEEE International Systems Conference (SysCon 2012), Vancouver, BC, Canada (March 2012)
27. Jhawar, R., Piuri, V., Santambrogio, M.: Fault tolerance management in cloud computing: A system-level perspective. IEEE Systems Journal 7(2), 288–297 (2013)
28. Juels, A., Kaliski, B.: PORs: Proofs of retrievability for large files. In: Proc. of the 14th ACM Conference on Computer and Communications Security (CCS 2007), Alexandria, VA, USA (October-November 2007)
29. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Authenticated index structures for aggregation queries. ACM Transactions on Information and System Security (TISSEC) 13(4), 32:1–32:35 (2010)
30. Li, J., Chen, X., Li, J., Jia, C., Ma, J., Lou, W.: Fine-grained access control system based on outsourced attribute-based encryption. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 592–609. Springer, Heidelberg (2013)
31. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. ACM Transactions on Storage (TOS) 2(2), 107–138 (2006)
32. Ostrovsky, R., Skeith III, W.E.: A survey of single-database private information retrieval: Techniques and applications. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 393–411. Springer, Heidelberg (2007)

33. Pang, H., Jain, A., Ramamritham, K., Tan, K.: Verifying completeness of relational query results in data publishing. In: Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), Baltimore, MA, USA (June 2005)
34. Samarati, P., De Capitani di Vimercati, S.: Data protection in outsourcing scenarios: Issues and directions. In: Proc. of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2010), Beijing, China (April 2010)
35. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: An extremely simple Oblivious RAM protocol. In: Proc. of the 20th ACM Conference on Computer and Communications Security (CCS 2013), Berlin, Germany (November 2013)
36. Wang, H., Yin, J., Perng, C., Yu, P.: Dual encryption for query integrity assurance. In: Proc. of the 2008 ACM International Conference on Information and Knowledge Management (CIKM 2008), Napa Valley, CA (October 2008)
37. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: Proc. of the 33rd International Conference on Very Large Data Bases (VLDB 2007), Vienna, Austria (September 2007)
38. Yang, Z., Gao, S., Xu, J., Choi, B.: Authentication of range query results in MapReduce environments. In: Proc. of the 3rd International Workshop on Cloud Data Management (CloudDB 2011), Glasgow, U.K. (October 2011)