Towards Owner-Controlled Data Sharing

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

Abstract We discuss some of the main problems related to allowing data owners to share their data with interested consumers in a controlled way in the context of digital data markets. Since resorting to the cloud for data storage reduces the burden at the owner's side, we first address the problem of supporting owners in selecting suitable cloud plans for storing their data collections. To this end, we illustrate some recent proposals for the specification and enforcement, in a friendly and flexible way, of the owners' requirements and preferences that should guide the selection process. We also address the problem of ensuring that data owners remain in control of their data and that they receive rewards for making their data available to others, and illustrate recent proposals addressing it.

1 Introduction

It goes without saying that data is the oil fueling a constantly growing number of activities and businesses of our society. Data are increasingly used to extract useful information and to create knowledge and predictions, generating an immense profit for the parties using them. Often, this happens without the involved users (i.e., the subjects *behind* the data themselves) being aware of their data being collected and used, and even more frequently without them being benefiting from the profits that directly derive from the usage of their data [1]. This has fostered a vision towards a fairer scenario, where individuals are first-class citizens and active participants of the data sharing ecosystem. An interesting direction concerns the development of *digital data markets*, where datasets can be traded between their owners and interested consumers (i.e., subjects wishing to access data) to generate knowledge and profits and where: *i)* sharing is controlled by the data owners (who can decide

Sabrina De Capitani di Vimercati · Sara Foresti · Giovanni Livraga · Pierangela Samarati
Università degli Studi di Milano, 20133 Milano, Italy
e-mail: {sabrina.decapitani, sara.foresti, giovanni.livraga, pierangela.samarati}@unimi.it

The final publication is available at Springer via https://doi.org/10.1007/978-3-030-87049-2_23

whether to share a piece of their data, and with whom); and *ii)* the owners get a reward for sharing their data with consumers (e.g., [2]). The awareness of the importance of these aspects is becoming more and more common among individuals. It is then easy to envision the possibility of such data markets being realized and possibly used.

In this context, the first aspect that owners should address concerns reasoning on how and where their data collections should be stored, for being easily (and selectively) available to interested data consumers. A possible approach for limiting the overhead at the owners' side consists in delegating data storage to external third parties. A natural possibility is then represented by outsourcing data collections to the cloud, leveraging one or more cloud plans available from the rich and diverse cloud market characterizing today's society. In this way, the overhead of storing and managing data is pushed from their owners to the chosen cloud providers, reducing owners' burden while, at the same time, enjoying the benefits of the cloud for making data available 24/7 from everywhere in the world. In this context, selecting the most suitable plan among the multitude available in the market is a critical problem, since different plans can exhibit different features and characteristics that can make them suitable to different application scenarios. Seemingly a non-critical and easy-to-solve problem, it is unfortunately far from being trivial. As a matter of fact, it requires a careful analysis of the features that can make the difference among plans, and a way to evaluate those features against possible requirements and preferences of users. It can also possibly require some technical skills or training, since the scenario is clearly characterized by cutting-edge ICT solutions. Selecting the right cloud plan hence represents a key factor for realizing an attractive digital data market: it is intuitive that if the market is built on cloud plans showing, for example, frequent downtimes or high latency, this would negatively impact the experience of all involved actors. In this chapter, we address such selection problem and illustrate recent approaches that can be adopted for cloud plan selection.

Once data have been moved to the cloud, they can be easily searched for and accessed by interested consumers. To ensure that owners maintain control over the sharing of their data, two aspects need to be considered. The first aspect, common to any scenario where a data owner resorts to external (and hence possibly not fully trusted) platforms for storing and managing data, consists in guaranteeing that each piece of data is shared with a subject (e.g., a consumer or the cloud provider) only with the will of its owner. This also requires to ensure that owners know, at any time and with high confidence, which consumer has had access to which portion of their data. A second aspect to be addressed, more specific to market scenarios, concerns the management of rewards to data owners for contributing their data. This is a critical aspect, especially when the interacting parties (the data owner, the cloud provider, and the interested data consumers) do not fully trust each other. In this chapter, we discuss these two aspects and illustrate recent approaches for solving them.

The reminder of this chapter is organized as follows. Section 2 addresses the problem of specifying requirements and selecting cloud plans for realizing a digital data market. It illustrates recent proposals based on a flexible specification language for formulating requirements, possibly using natural language expressions. Section 3 focuses on the issue of ensuring owner-controlled sharing of data. It illustrates a recent proposal building on selective owner-side encryption to ensure that owners remain in control of who can access which portions of their data collections. Section 4 addresses the problem of the management of rewards to data owners. It illustrates a possible solution building on blockchain and smart contracts to ensure that owners receive an agreed reward whenever an interested consumer gets access to some of their data. Finally, Section 5 concludes this chapter.

2 Cloud plan selection

A first problem to be addressed when moving data collections to the cloud concerns the selection of the most suitable set of cloud plans for data storage. This demands the evaluation of the features of the different candidate cloud plans with respect to possible requirements and preferences the owner could have, in turn demanding for approaches supporting owners in formulating such requirements. In this section, we first discuss some basic concepts and approaches addressing this problem (Section 2.1). We then illustrate recent solutions that permit to specify and enforce arbitrary requirements and preferences (Section 2.2), possibly leveraging natural language expressions (Section 2.3).

2.1 Basic concepts

The first aspect to address in our scenario concerns determining the features, among those characterizing the different plans, which are relevant to the needs of the owner, and which can be used to drive the selection process. As an example, intuitive features can encompass the performance or the availability characterizing the different plans [3, 4]. Traditional proposals for cloud plan selection are based on Quality-of-Service (QoS) attributes that are guaranteed in Service Level Agreements (SLAs) by the providers for their services (e.g., [5, 6, 7]). In this regard, the scientific community has proposed several solutions for selection. These include the evaluation of low-level characteristics (such as CPU and network throughput) and cost of available plans (e.g., [8]), the definition of standardized bodies of more complex QoS attributes (e.g., [9]), the combination of QoS evaluation and other criteria such as subjective assessment and personal/past experience (e.g., [10, 11, 12, 13]). It is interesting to note that some approaches have suggested to evaluate the *user-side* QoS rather than the *provider-side* QoS, meaning the values of QoS attributes measured at the user side (i.e., at the owner) rather than those declared by providers (e.g., [14]). Also, specific approaches have put forward the idea of considering security-related attributes in the selection (e.g., [15, 16, 17, 18]).

Recently, the scientific community has started to investigate the possibility of supporting owners in arbitrarily selecting the features and characteristics over which

	P_1	P_2	P_3	P_4	
prov	provA	provA	provA	provB	*cloud provider*
loc	locB	locA	locB	locB	*geographical location of servers*
encr	AES	AES	AES	3DES	*adopted encryption*
band	25	25	20	15	*bandwidth (Gb/s)*
test	med	top	top	low	*penetration test authority*
cert	certC	certA	certC	certB	*security certification*
aud	—	—	—	1Y	*security auditing frequency*

Fig. 1 Abstract representation of cloud plans

their requirements should be defined (e.g., [19, 20]). This paradigm shift is typically based on the existence of a broker [19], that is, an entity in charge of collecting and understanding such arbitrary requirements and assessing their satisfaction by the candidate plans. By permitting the definition of arbitrary requirements on arbitrary features (clearly, representing characteristics that can be evaluated), such proposals allow for greater flexibility and user-friendliness than traditional approaches that are limited to a pre-defined set of features. In the remainder of this section, we illustrate two recent proposals that pursue this direction by allowing owners to leverage a friendly specification language [19] and natural language expressions [21] to formulate arbitrary requirements and preferences.

2.2 Crisp requirements and preferences

In this section, we illustrate how it is possible for data owners to easily formulate requirements to guide the selection of the cloud plan that best fits their needs. We will focus on the framework in [19], since it proposes a flexible and user-friendly language for supporting the specification of both *requirements* and *preferences* (i.e., hard and soft constraints, respectively) that can then be used to identify those plans that can be considered acceptable (requirements) and, among the acceptable ones, the ones that are preferable (preferences). As mentioned in Section 2.1, the proposal in [19] permits to formulate requirements and properties over arbitrary *attributes* of interest (configuration parameters) such as those appearing in the SLAs of the different plans, as well as metadata associated with them (e.g., the provider of a plan, or the nationality of the provider). To this end, plans are represented as vectors with one element for each attribute of interest reporting the value that such attribute assumes for the plan (or the special value '—' if the value of such attribute is unknown/unspecified for the plan). Figure 1 presents an example of such vectors for four plans P_1, P_2, P_3, and P_4, defined over seven different attributes. For instance, P_1[prov] = provA means that plan P_1 is offered by provider provA. P_1[aud] = — means that the frequency of security auditing for P_1 is not specified/unknown.

Complex requirement	**Semantics**
ANY($b_1, \ldots, b_n$)	alternatives among base requirements
ALL($b_1, \ldots, b_n$)	sets of base requirements to be jointly satisfied
IF ALL($b_1, \ldots, b_k$) THEN ANY($b_{k+1}, \ldots, b_n$)	conditional requirements
FORBIDDEN($b_1, \ldots, b_n$)	forbidden configurations
AT_LEAST(m, ($b_1, \ldots, b_n$))	at least m base requirements ($m < n$)
AT_MOST(m, ($b_1, \ldots, b_n$))	at most m base requirements ($m < n$)

(a)

b_1 :	`prov`(provA, provB, provC)
b_2 :	¬`band`(0.2, 5)
c_1 :	ALL({`loc`(locA, locB), ¬`encr`(DES)})
c_2 :	ANY({`test`(top, med), `cert`(certA, certB)})
c_3 :	ANY({`loc`(locA), `cert`(certC)})
c_4 :	IF ALL({`loc`(locB), `encr`(3DES)) THEN ANY(`aud`(3M, 6M), `cert`(certA))
c_5 :	IF ALL(`test`(—)) THEN ANY(`cert`(certA))
c_6 :	FORBIDDEN({¬`loc`(locA), `test`(low)})
c_7 :	AT_MOST(2, {`prov`(provB), `band`(15), `encr`(3DES)})
c_8 :	AT_LEAST(2, {`loc`(locA), `encr`(AES), `prov`(provA, provB)})

(b)

Fig. 2 Complex requirements supported by the language in [19] (a) and an example of a set of requirements over the plans in Figure 1 (b)

2.2.1 Requirements and preferences specification

The proposal in [19] builds on the idea that requirements and preferences should identify the values of the attributes that are considered mandatory or preferable for a plan to satisfy the needs of the data owner. The building block for the definition of requirements is the concept of *base requirement*, which is denoted by b. Given a set $\{v_1, \ldots, v_n\}$ of values in the domain of an attribute `attr`, a base requirement on `attr` imposes that `attr` can assume (`attr`($v_1, \ldots, v_n$)) or cannot assume (¬`attr`($v_1, \ldots, v_n$)) such a set of values. For instance, a base requirement of the form `prov`(provA, provB, provC) states that, to be acceptable, a plan must be offered by provider provA, provB, or provC.

Starting from base requirements, the specification language in [19] permits to express a variety of *complex requirements,* summarized in Figure 2(a). An ANY requirement models alternatives among base requirements, and demands that at least one base requirement in the set $\{b_1, \ldots, b_n\}$ be satisfied. Similarly, an ALL requirement demands that all the listed base requirements be satisfied. A conditional IF-THEN requirement demands that if all the requirements appearing in the IF part are satisfied, then at least one of those in the THEN part must be satisfied. A FORBIDDEN requirement demands that the involved requirements must not be satisfied together in a plan, since they represent a forbidden configuration. An AT_LEAST (AT_MOST, respectively) requirement demands that at least (at most, respectively) m out of the n related base requirements be satisfied. Figure 2(b) illustrates an example of a set of base (b_1 and b_2) and complex ($c_1, \ldots, c_8$) requirements specified over the plans in Figure 1. For instance, base requirements b_1 and b_2, respectively, demand

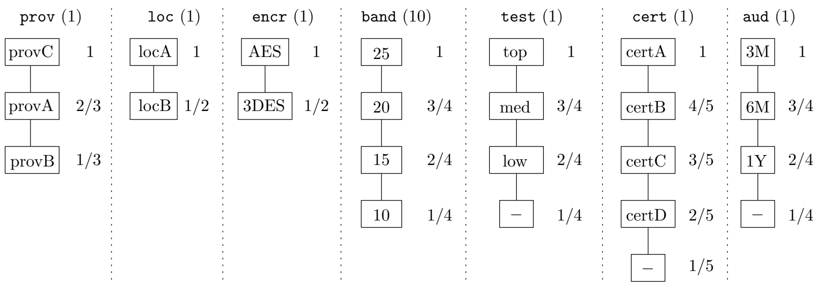


Fig. 3 Preferences for the plans in Figure 1

that an acceptable plan must be offered by provider provA, provB, or provC (b_1), and must not have a bandwidth equal to 0.2 Gb/s or 5 Gb/s (b_2). Complex ALL requirement c_1 states that a plan can be considered acceptable only if its servers are geographically located in locA or locB, and if the adopted encryption is different from DES. Complex FORBIDDEN requirement c_6 states that a plan whose servers are not located in locA and for which the penetration test is enforced by the authority named ‘low’ cannot be considered acceptable.

As for the preferences, the framework in [19] permits two levels of specification, on the attribute values and on the attributes themselves. In particular, preferences on attribute values model the fact that, for a certain attribute, some values are preferred over other ones. A natural and intuitive interpretation can then model such preferences as a total order relationship among sets of values that the different attributes can assume. In other words, specifying preferences on attribute values can be done by first partitioning the values that an attribute can assume (indeed, those that are acceptable according to the requirements) in sets of equivalently-preferred values, and then specifying an order relationship among these value sets. Such kind of preference can be graphically represented as a hierarchy where, for example, preferred elements appear higher than less preferred ones. Figure 3 illustrates an example of preferences over attribute values for the plans in Figure 1. Here, for example, value locA is preferred over value locB for attribute `loc`. The figure also reports, for each value, its *score* (used in the enforcement phase) given by its relative position in the induced ranking. Given the number h of partitions in which the values are grouped, the least preferred value(s) has score of $1/h$. Going up in the hierarchy, scores increase of $1/h$ at each step. The most preferred value(s) has then a score equal to 1. To illustrate, consider attribute `prov` in Figure 3: its values have been partitioned in three sets ($h = 3$), and hence value provB (the least preferred) has score of $1/h = 1/3$. Value provA, immediately better than provB, has score $1/3$ higher than that of provB, and hence equal to $1/3 + 1/3 = 2/3$. Value provC, immediately better than provA, has score $1/3$ higher than that of provA, and hence equal to $2/3 + 1/3 = 1$.

Preferences on attributes, on the other hand, can be used to specify the relative importance that the data owner assigns to the attributes. The proposal in [19]

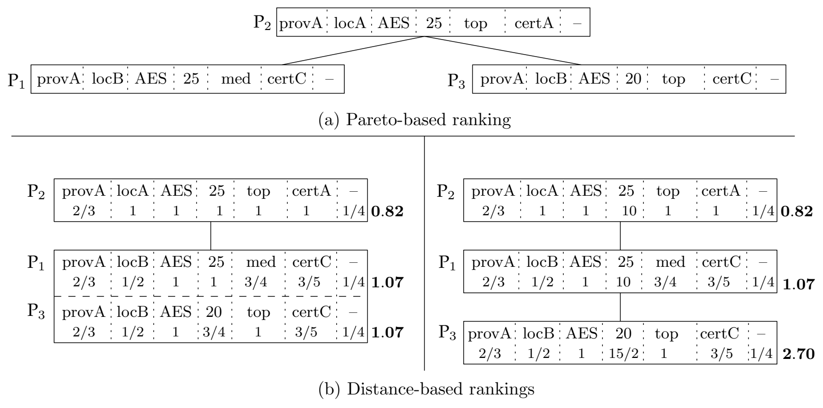

P2 provA locA AES 25 top certA –
P1 provA locB AES 25 med certC –
P3 provA locB AES 20 top certC –
(a) Pareto-based ranking
P2 provA locA AES 25 top certA – 2/3 1 1 1 1 1 1/4 0.82
P1 provA locB AES 25 med certC – 2/3 1/2 1 1 3/4 3/5 1/4 1.07
P3 provA locB AES 20 top certC – 2/3 1/2 1 3/4 1 3/5 1/4 1.07
P2 provA locA AES 25 top certA – 2/3 1 1 10 1 1 1/4 0.82
P1 provA locB AES 25 med certC – 2/3 1/2 1 10 3/4 3/5 1/4 1.07
P3 provA locB AES 20 top certC – 2/3 1/2 1 15/2 1 3/5 1/4 2.70
(b) Distance-based rankings

Fig. 4 Rankings of plans P_1, P_2, and P_3 in Figure 1 according to the preferences in Figure 3

models them through a weight function assigning higher weights to more important attributes. Figure 3 illustrates also an example of weights assigned to the attributes of our running example, reported as the number appearing in brackets close to the name of the attributes. In this example, all attributes have weight 1, except attribute `band`, which has weight 10.

2.2.2 Requirements and preferences evaluation

As mentioned above, the framework in [19] leverages requirements to determine *acceptable plans*, which are then ranked according to how much they respond to preferences.

A plan can be considered acceptable if and only if it satisfies *all* the stated requirements. The approach in [19] for determining the acceptable plans is based on a Boolean interpretation of the requirements, which also offers the possibility of checking whether the overall set of requirements is satisfiable or there are conflicting requirements (e.g., a requirement demands a certain value for an attribute, and another requirement excludes that value from those acceptable). With reference to the plans in Figure 1 and the requirements in Figure 2(b), it is easy to see that plans P_1, P_2, and P_3 are acceptable, while P_4 is not (since it does not satisfy requirements c_3, c_4, c_6, c_7, c_8).

As for preferences, the solution in [19] proposes different approaches for ranking plans. The first, and more intuitive, is a Pareto-based ranking, where a plan P_a is ranked better then a plan P_b iff P_a shows values that are equally or more preferred than those showed by P_b for all attributes and, for at least one attribute, P_a shows a value that is preferred than that showed by P_b. For instance, as illustrated in Figure 4(a), plan P_2 Pareto-dominates (and is hence preferable to) P_1 and P_3, which are instead not comparable between them. To overcome the possibility of returning incomparable solutions, a distance-based approach can be used instead of the Pareto-based ranking. The distance-based approach is based on the notion of *ideal plan*, a plan that shows the most preferred values for all attributes. The 'closer' a plan is to such an ideal plan, the better it satisfies the stated preferences. Plans are then modeled as points in an m-dimensional space, with m the number of attributes characterizing plans. Given a plan, its position in the space is given by a set of coordinates being the scores assigned to its attribute values based on the ranking induced by the preferences. For instance, plan P_2 has coordinates [2/3, 1, 1, 1, 1, 1, 1/4] given by the scores of its attribute values in the ranking in Figure 3 (e.g., 2/3 is the score associated with value P_2[prov] = provA). Clearly, the values of the coordinates of the ideal plan will be, by definition, 1 for all dimensions. Leveraging such interpretation of plans it is then immediate, for example through a simple computation of the Euclidean distance, to evaluate how close the plans are to the ideal plan. Such distance-based approach also permits to consider the preferences on attributes, by simply weighting the different elements in the coordinates of a plan by the weight given to the corresponding attribute. Figure 4(b) graphically illustrates the distance-based rankings over the acceptable plans of our running example (where the figure on the right-hand side considers attribute weights). In the figure, the distance of each plan to the ideal plan is reported in boldface on the right-hand side of each plan in the ranking.

2.3 Requirements with natural language

The approach illustrated in Section 2.2 provides data owners with a flexible and user-friendly language for specifying arbitrary requirements and preferences. However, such approach can still show the complication, intrinsic to the definition of crisp and precise requirements, of demanding domain-specific technical knowledge to fully understand the attributes characterizing cloud plans and the meaning of their values. As a matter of fact, requirement evaluation crosses out plans simply based on acceptable/unacceptable values. Whenever the owner formulating requirements is not technically skilled or trained, this may represent a barrier to requirement specification, ultimately leading to non-optimal solutions. Intuitively, it would be easier for non-skilled owners to specify their requirements using linguistic labels (such as 'high' or 'low' or 'important') instead of crisp values, and possibly on high-level properties (such as 'security', or 'performance') instead of on attributes representing low-level (configuration) parameters such as those in SLAs. In this section, we present how it is possible to support owners in an easy and intuitive requirement specification, leveraging natural language and high-level properties [21].

2.3.1 Requirements specification

The proposal in [21] builds on the definition of *abstract parameters* and *abstract concepts*. Abstract parameters permit owners to specify their requirements over low-level configuration parameters (e.g., attribute `bandwidth` used in the example for the framework in Section 2.2) through natural language expressions. For instance, an owner could state a requirement for 'high' `bandwidth`, without specifying a precise threshold and using instead the linguistic label 'high'. Abstract concepts complement abstract parameters by representing higher-level abstractions over parameters. For instance, an owner could state a requirement over `performance`, which intuitively represents an abstraction over lower-level parameters `throughput` and `bandwidth`. Owners (and especially non-skilled or non-trained ones) can then operate on abstract parameters and concepts to specify in a friendly and flexible manner their desiderata.

Operating with abstract parameters and concepts requires the existence of a set of *linguistic labels* that can be associated with them as values (in contrast to domain-specific values). Such sets can be arbitrarily defined in such a way to 'quantify' a parameter or a concept. As an example, abstract parameter `bandwidth` could be associated with a set {small, large} of linguistic labels. Similarly, abstract concept `performance` could be associated with a set {low, med, high} of linguistic labels.

As already mentioned, abstract concepts and parameters are clearly strictly connected, since concepts represent abstractions over parameters, more easily accessible by non-skilled owners. For instance, with reference to the example above, a high `throughput` and a large `bandwidth` can result in a high `performance` value. Following this intuition, the relationship between concepts and the involved parameters is modeled through a set of *implication rules*, specifying conditions on which combination of values (linguistic labels) for parameters imply a given value (linguistic label) for concepts. To avoid ambiguities, each label associated with a concept should be regulated by an implication rule. For instance, since concept `performance` is associated (in the examples above) with three linguistic labels, it is governed by three inference rules:

$\langle \texttt{throughput} = \mathsf{high} \rangle \vee \langle \texttt{bandwidth} = \mathsf{large} \rangle \implies \langle \texttt{performance} = \mathsf{high} \rangle;$
$\langle \texttt{throughput} = \mathsf{med} \rangle \implies \langle \texttt{performance} = \mathsf{med} \rangle;$
$\langle \texttt{throughput} = \mathsf{low} \rangle \vee \langle \texttt{bandwidth} = \mathsf{small} \rangle \implies \langle \texttt{performance} = \mathsf{low} \rangle.$

Equipped with a vocabulary for reasoning over parameters and concepts with linguistic labels, it is then possible to formulate requirements. Intuitively, in line with the 'quantitative' approach permitted for parameters and concepts, requirements represent how much a certain combination of values (linguistic labels) for the abstract parameters and concepts satisfies the owner formulating them. The rationale is to support the specification of rules that say that a certain combination of characteristics is, for example, *highly satisfactory*, while another combination is *not satisfactory*. Requirements are then modeled as the implication rules abovementioned: a combination of linguistic expressions over a set of parameters and concepts implies a certain linguistic expression (e.g., 'low' or 'high') of ad-hoc variable `satisfaction`, modeling the overall satisfaction for a plan. For instance,

"$\langle$`performance` = high$\rangle \Longrightarrow \langle$`satisfaction` = high$\rangle$" is an example of a requirement, defined over concept `performance`, stating that a high level of performance highly satisfies the owner.

2.3.2 Requirements evaluation

To evaluate the specified requirements, it is necessary to establish a mapping between the elements included in the requirements (i.e., expressions using natural language and possibly referred to abstract concepts) to the actual (crisp) characteristics of the different plans. This requires to map: *i)* concepts to actual configuration parameters; and *ii)* linguistic labels to crisp (domain-specific) values. An intuitive approach for this second problem could involve associating with a certain linguistic label a set of crisp values. For instance, given the set of crisp values that can be assumed by parameter `bandwidth` (e.g., [0.2Gb/s, 25Gb/s]), one may partition such set in disjoint intervals (e.g., [0.2Gb/s, 10Gb/s), [10Gb/s, 25Gb/s]), and associate each interval with a linguistic label (e.g., [0.2Gb/s, 10Gb/s) with small and [10Gb/s, 25Gb/s] with large). This solution, while indeed viable, carries the drawback of creating sharp boundaries among the sets of values that correspond to the different linguistic labels. With reference to the example above, a sharp boundary is created around value 10 Gb/s: a value of 9.99Gb/s would be considered small, while the slightly larger value 10Gb/s would be considered large. A more flexible approach, which enjoys the advantage of also providing with a means to manage the correspondence between abstract concepts and actual parameters, can be based on the adoption of fuzzy logic [21]. Abstract parameters and concepts can be interpreted as *fuzzy variables*, and the adopted linguistic labels as *fuzzy sets*. Fuzzy variables are variables that can assume crisp values and linguistic labels, while fuzzy sets are sets whose elements have a degree of membership, expressed as a value in the continuous interval [0, 1]. Given an element, in classical set theory it either belongs or does not belong to a set. In fuzzy set theory, an element belongs to a fuzzy set with a certain *degree of membership* (and of course can belong to different fuzzy sets, possibly with different degrees of membership). The degree of membership μ of elements to a fuzzy set is governed by the definition of *membership functions*, which can assume different shapes and permit a gradual assessment of the membership of elements to sets. Figure 5(a) illustrates an example of membership functions that can be associated with parameter `bandwidth`, one for each linguistic label that can be associated with it (i.e., small and large). As it can be seen in the figure, the functions operate over the domain of crisp values that can be assumed by the parameter, and define how much a certain value belongs to the fuzzy set represented by the linguistic labels, meaning how much a certain value is 'representative' of each linguistic label. For instance, consider a value x in the domain [0.2, 25] of `bandwidth` and the membership function regulating label large: it is immediate to see that the more value x grows, the more it belongs (i.e., the higher its degree μ of membership) to the fuzzy set large. Similarly, considering the membership function regulating label small, the more value x grows, the less it belongs (i.e., the lower its degree μ of membership) to the fuzzy set small.

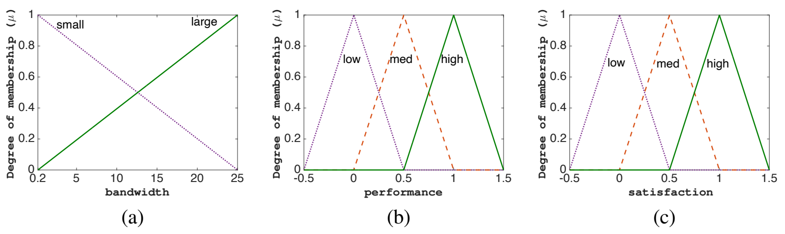


Fig. 5 An example of membership functions for parameter `bandwidth` (a), concept `performance` (b) and the ad-hoc variable `satisfaction` (c)

Abstract parameters have then a natural interpretation in terms of fuzzy logic, establishing a correspondence (through membership functions) between the linguistic labels adopted in the requirements and the original crisp values assumed by a configuration parameter. A similar approach can be used also for managing abstract concepts and the variable `satisfaction` used in the requirements with the note that, being abstractions, they do not have a natural domain of crisp values. An intuitive approach is then to arbitrarily define their domains, for example in the continuous interval $[0, 1]$, and then again establish a mapping through membership functions. Figures 5(b)–(c) illustrate examples of membership functions for the abstract concept `performance` and for the ad-hoc variable `satisfaction`.

To quantify concepts and the `satisfaction` of the owner as well as to reason about and evaluate the requirements, the proposal in [21] uses *fuzzy logic* and, more precisely, *fuzzy inferences*. In a nutshell, a fuzzy inference process takes as input a (set of) crisp value(s), interprets it (them) with a fuzzy modeling as illustrated before, evaluates a set of if-then rules based on such fuzzy modeling obtaining a (fuzzy) result, and returns such result after having transformed it again into a crisp value. Fuzzy inference can then easily provide the framework in which evaluating requirements: a first inference process leverages the implication rules linking concepts and parameters as the if-then rules of the process. It takes as input the crisp parameter values characterizing the plans under analysis and quantifies the concepts used in the requirements. A second inference process leverages instead the requirements themselves as if-then rules. It takes as input the quantification of the concepts done in the first inference (and, if present in the requirements, also the crisp values for parameters) and quantifies the level of `satisfaction`.

To illustrate the working of the fuzzy inference process, consider requirement "$\langle$`performance` = high$\rangle \Longrightarrow \langle$`satisfaction` = high$\rangle$", taken from a set of requirements, and suppose that concept `performance` depends on low-level parameters `throughput` and `bandwidth`. Clearly, candidate plans exhibit values (e.g., in their SLAs) for `throughput` and `bandwidth` but not for `performance`. The framework in [21] would then apply a first inference process as follows: *i)* the crisp values for `throughput` and `bandwidth` are taken as input; *ii)* the inference rules governing concept `performance` are used as if-then rules and applied to the (fuzzified) input values; *iii)* depending on the input values and on the evaluated rules, a quantification

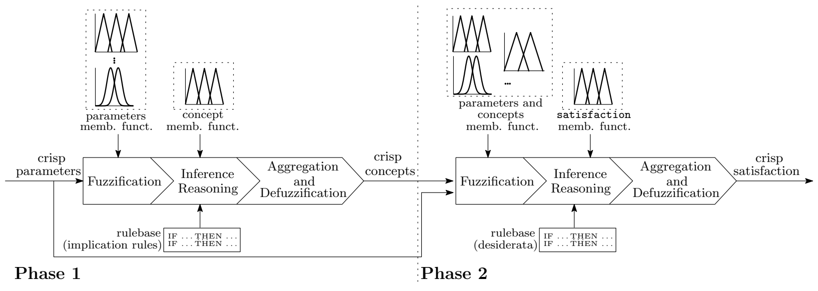


Fig. 6 Fuzzy inference processes for evaluating requirements[1]

of `performance` is then produced. With such value for `performance`, a second inference process would then operate to assess owner's satisfaction. The second inference process operates exactly as the first one, with the difference that it takes as input the quantification of `performance` and operates on it with the rules expressed in the requirements. Figure 6 graphically illustrates this two-steps approach. We close this section with the note that the inference processes require the definition of different domain-specific operations, such as the fuzzification of crisp values and the defuzzification of fuzzy values, which can be performed adopting different existing approaches.

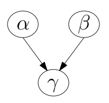
3 Controlled sharing

While moving data to the cloud reduces the overhead left at the owners' side, it also causes the owners to lose direct control over their data and on who can access them [22]. To ensure a fair data market, it is crucial to empower the owners with control over the sharing of their data, and to guarantee that they receive rewards for making their data available to others (which is addressed in Section 4). In this section, after briefly recalling some basic concepts (Section 3.1), we illustrate a recent approach enabling controlled sharing. This approach allows owners to share their data (to which, for the sake of generality, to which we refer with the term 'resources') to interested consumers ensuring that owners remain in control of who can access which resources (Section 3.2).

3.1 Building blocks

The solution in [23] ensures controlled data sharing by leveraging two main building blocks: *i)* selective owner-side encryption and *ii)* key derivation.

[1] ©2019 IEEE. Reprinted, with permission, from [21].



(a)

$\mathcal{T}$

From	To	Value
α	γ	$k_\gamma \oplus \mathcal{H}(k_\alpha, l_\gamma)$
β	γ	$k_\gamma \oplus \mathcal{H}(k_\beta, l_\gamma)$

(b)

Fig. 7 An example of key derivation structure (a) and token catalog (b)

Selective owner-side encryption. Selective owner-side encryption protects the confidentiality of the resources to which it is applied through encryption. It is particularly appealing in scenarios of selective sharing as it consists in encrypting, at the owner-side, different resources with different keys. Keys are then distributed to consumers in such a way that each consumer can decrypt all and only the resources she is authorized to access. Since the encryption layer is applied at the owner side, resources self-enforce the access restrictions defined over them. Also, owner-side encrypted resources are protected against the cloud provider storing them as, without encryption keys, it cannot perform decryption. This represents a convenient feature in the considered scenario, since the owner can enjoy the benefits of resorting to the cloud for data storage while resting assured that only consumers knowing the encryption keys will be able to decrypt resources. A straightforward solution to enforce access restrictions through selective encryption consists in encrypting each resource with a different key, and in distributing to each consumer the keys of the resources she can access. However, this practice would imply a considerable key management burden for owners and consumers. To mitigate such overhead, the proposal in [23] leverages key derivation.

Key derivation. Key derivation permits to derive an encryption key k_y from the knowledge of another encryption key k_x and of a public label l_y (i.e., a piece of information) associated with k_y [24, 25]. The derivation of k_y from k_x is enabled by a public token $t_{x,y}$ computed as $k_y \oplus \mathcal{H}(k_x, l_y)$, with $\oplus$ the bitwise `xor` operator, and $\mathcal{H}$ a cryptographic hash function. The derivation of k_y from k_x can be *direct*, leveraging a single token $t_{x,y}$, or *indirect*, through a chain $\langle t_{x,z_1}, \ldots, t_{z_n,y} \rangle$ of tokens. Key derivation structures can be graphically represented as directed acyclic graphs, where vertices represent encryption keys (and their labels), and edges represent tokens among them. Since tokens do not need to be kept private, they can be physically stored in a public catalog $\mathcal{T}$. Figure 7 illustrates an example of derivation among three keys k_α, k_β, and k_γ (Figure 7(a)) and the corresponding token catalog $\mathcal{T}$ (Figure 7(b)) . For simplicity, in our examples, we use x to denote the label of key k_x and, in the figures, we use the label x of key k_x to denote the corresponding vertex v_x (e.g., vertex v_α, denoted α, in Figure 7 represents key k_α and its label α). In the following, when clear from the context, we will use the terms keys and vertices (tokens and edges, respectively) interchangeably.

3.2 Ensuring controlled sharing

We now illustrate how selective owner-side encryption and key derivation can be used to effectively enforce the authorization policy specified by the data owner, hence providing for owner-controlled sharing in the data market.

In principle, the authorization policy can be represented in different ways, including access control lists (reporting, for each resource, the list of consumers authorized for access) as well as capability lists (reporting, for each consumer, the list of authorized resources). Given the dynamic scenario considered, characterized by consumers leveraging the market to purchase sets of resources, it is natural to think in terms of sets of resources and hence to represent authorizations as capability lists. Given a consumer c, $\mathsf{cap}(c)$ represents the set of resources for which c has purchased access (see Section 4). The capability $\mathsf{cap}(c)$ is then updated whenever c purchases access to a new resource r and is then authorized for r (i.e., $\mathsf{cap}(c)$:=$\mathsf{cap}(c)\cup\{r\}$).

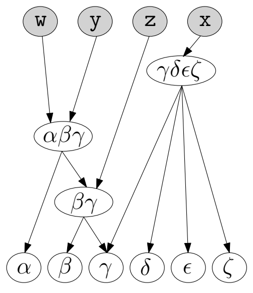
Ensuring controlled sharing requires that the content of the resources remains protected and only authorized consumers can access it. Moreover, despite resorting to the cloud for storage and hence losing direct control over their resources, the data owners must be aware, at all times, of which consumers have access to which resources. Selective owner-side encryption (Section 3.1) represents a promising solution that enjoys the advantage of maintaining resources confidential also to the cloud provider. We will now illustrate how sharing of resources can be controlled.

With selective owner-side encryption, the owner needs to agree a key k_c with every consumer c with which she wants to share resources. To grant c access to the resources in her capability list $\mathsf{cap}(c)$, the owner publishes a set of tokens enabling the derivation of the keys used to encrypt the resources in $\mathsf{cap}(c)$ starting from k_c. A (basic) key derivation structure includes a vertex v_c for each consumer c (representing k_c, which is known to c), a vertex v_r for each resource r (representing k_r, which is used to encrypt r), and a set of edges (representing tokens for derivation) connecting, for each consumer c, vertex v_c to vertex v_r for each $r \in \mathsf{cap}(c)$. This means that it is possible to derive, from key k_c, the encryption key k_r for each resource r in $\mathsf{cap}(c)$. To keep the size of the token catalog under control, the key derivation structure can be enriched with additional vertices representing keys used for derivation purposes only. In line with the authorization policy being represented as capability lists, such additional vertices represent sets of resources [23]. A possible approach for correctly enforcing an authorization policy in a market consists in connecting vertices through edges (i.e., keys through tokens) in such a way that [23]:

- for each consumer c, vertex v_c is connected to vertex $v_{\mathsf{cap}(c)}$ representing the set of resources in her capability list;
- each vertex v_{R_i}, representing a set R_i of resources, is directly connected to other vertices according to the subset containment relationship (i.e., for each edge (v_{R_i}, v_{R_j}), $R_i \supset R_j$) and in such a way that set R_i is fully covered by the resources represented by these other vertices;
- for each consumer c, there is a path connecting vertex v_c to all vertices v_r such that $r \in \mathsf{cap}(c)$.

Consumer c	cap(c)
w	α, β, γ
x	$\gamma, \delta, \epsilon, \zeta$
y	α, β, γ
z	β, γ

(a)



(b)

Fig. 8 An example of an authorization policy for four consumers and six resources (a), and of a key derivation structure that enforces it (b)

Figure 8 illustrates an example of an authorization policy regulating access to a set $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\}$ of six resources for four consumers w, x, y, and z (Figure 8(a)), and of a key derivation structure that enforces it (Figure 8(b)). The structure contains a vertex for each consumer (gray vertices), representing the key agreed between the owner and the consumer (and known to the consumer herself), and a vertex for each resource. It also contains three additional vertices $\beta\gamma$, $\alpha\beta\gamma$, and $\gamma\delta\epsilon\zeta$ representing the capability lists of the consumers. Vertices of consumers are directly connected to the vertices of their capability lists, which are in turn connected to other vertices respecting the subset containment relationship and such that they are covered (e.g., vertex $\alpha\beta\gamma$ is connected to vertices α and $\beta\gamma$, in turn connected to vertices β and γ). The derivation structure correctly enforces the authorization policy in the figure, since there is a path linking each consumer vertex to all and only the vertices representing the resources in her capability list. For instance, consider consumer w with cap(w) $= \{\alpha, \beta, \gamma\}$. From her key $k_{\mathtt{w}}$, she can use token $t_{\mathtt{w},\alpha\beta\gamma}$ to derive $k_{\alpha\beta\gamma}$ (note that this key is used for derivation purposes only and does not encrypt any resource). From $k_{\alpha\beta\gamma}$, she can then use token $t_{\alpha\beta\gamma,\alpha}$ to derive key k_α, and token $t_{\alpha\beta\gamma,\beta\gamma}$ to derive key $k_{\beta\gamma}$, from which she can then use tokens $t_{\beta\gamma,\beta}$ and $t_{\beta\gamma,\gamma}$ to derive k_β and k_γ, respectively. Hence, starting from the knowledge of a single key $k_{\mathtt{w}}$, w can derive the keys of all the resources shared with her.

Having illustrated how the key derivation structure can be built, we now illustrate how it can be updated to reflect changes to the authorization policy. The structure must be updated when: *i)* new resources are placed in the market; and *ii)* new accesses are requested by (and granted to) consumers. We note that authorization revocation is not in line with the fact that access is provided upon payment [23].

The insertion of a new resource r into the market is reflected in the key derivation structure by simply adding a vertex v_r. This requires the owner to generate an encryption key k_r and a label l_r, encrypt r with k_r, and place the encrypted version of r in the market (i.e., outsource r to the chosen cloud platform). For instance, with reference to the structure in Figure 8(b) and assuming that the owner publishes all six resources before any consumer request, the derivation structure includes only the leaves of the structure.

The purchase of a set R of resources by consumer c is reflected in the key derivation structure by ensuring that there is a path from vertex v_c of the consumer c to the vertices of all the resources in the updated capability list $\mathsf{cap}(c)$=$\mathsf{cap}(c)\cup R$. It is first necessary to create, if not already in the structure, vertices v_c for c and $v_{\mathsf{cap}(c)}$ for the updated capability list of c (creating also the corresponding encryption keys and labels). Vertex v_c is then connected to vertex $v_{\mathsf{cap}(c)}$ that, as illustrated previously, is connected to the other vertices in the hierarchy following the subset containment relationship and coverage principle. Each added edge corresponds to a token, which is inserted into the catalog. Figure 8(b) illustrates a possible key derivation structure after a series of granted requests, allowing the consumers to access resources according to the policy in Figure 8(a).

4 Rewards to owners

In this section, we illustrate a possible approach for ensuring that owners receive a reward every time they grant access to some of their resources to interested consumers. This approach nicely complements the selective owner-side encryption approach illustrated in Section 3.

Ensuring rewards is a complex aspect in the addressed scenario, for two main reasons: *i)* consumers and owners might not completely trust each other, and *ii)* both parties could in principle misbehave to obtain illicit benefits. The main misbehaviors that may happen are related to the payment of a reward (i.e., a malicious consumer does not pay the reward for a resource she accessed, or a malicious owner falsely claims that a reward has not been paid, demanding a new payment) as well as to the access to a resource (where a malicious owner does not grant access despite having received a payment, or a malicious consumer falsely claims that access has not been granted despite the payment, requesting her money back). All misbehaviors could hamper the adoption of markets to trade (personal) data, disincentivizing owners to contribute with their data. In this section, after a discussion of some basic concepts (Section 4.1), we illustrate how the solution in [23] ensures rewards to owners, while preventing possible misbehaviors (Section 4.2).

4.1 Building blocks

The solution in [23] ensures rewards to owners leveraging two main building blocks: *i)* blockchain and *ii)* smart contracts.

Blockchain. A blockchain is a shared and public ledger of transactions organized as a list of blocks, linked in chronological order [26]. Each block contains a certain number of transaction records as well as a cryptographic hash of the previous block.

The blockchain is then maintained in a distributed way by a decentralized network of peers. While the single peers might not trust each other, the content and status of the blockchain is continuously agreed upon since each transaction is validated by the network of peers, and is then included in a block through a consensus protocol. A peculiarity of blockchains, which provide trust even if the peers singularly taken are not fully trusted, is that everyone can inspect a blockchain but no single peer can tamper with it, since modifications to the content of a blockchain requires mutual agreement among peers. Consequently, nobody can modify a committed block, and possible updates are reflected in a new block containing the updated information. This permits to trust the content and the status of a blockchain, while not trusting the single peers.

Smart contracts. Smart contracts can be used to establish an agreement among multiple, possibly distrusting, parties through a blockchain. A smart contract is a piece of software deployed on a blockchain, and is typically composed of a set of rules on which the interacting parties have to agree. The rules are of the form 'if-then' and define events and subsequent actions. Such rules formalize the clauses of the contract to be agreed upon (and virtually signed) by the parties. By leveraging the underlying blockchain consensus protocol, the execution of a smart contract can be trusted for correctness, meaning that all the conditions of the agreement have been met (as validated by the network). However, smart contracts and their execution do not provide confidentiality and privacy guarantees, as open visibility over the content of a contract and over the data it manipulates is a necessary condition for validation [27].

4.2 Ensuring rewards to owners

The possibility of consumers not paying for a granted resource (and conversely the possibility of a malicious owner claiming that she did not receive a payment for a resource, while she actually has) can be easily prevented adopting blockchain and smart contracts. The basic idea consists in ensuring that access to a resource be granted *upon* a monetary transaction (i.e., the payment of the reward) occurring between the owner and the consumer purchasing access. In this way, the money transfer can be inspected and validated by the blockchain network. Considering that access is granted through the possibility of decrypting resources, two straightforward approaches could be envisioned. A first approach could directly trade encryption keys through a smart contract. A second approach could trigger the updates to the derivation hierarchy (see Section 3.2) to ensure access directly through the smart contract, computing and communicating tokens within the smart contract. However, neither of these approaches is feasible since, as mentioned above, smart contracts are public and hence their simple observation would disclose the traded secrets (i.e., the encryption keys in both approaches, since also updating the derivation structure requires knowledge of the keys used in the system).

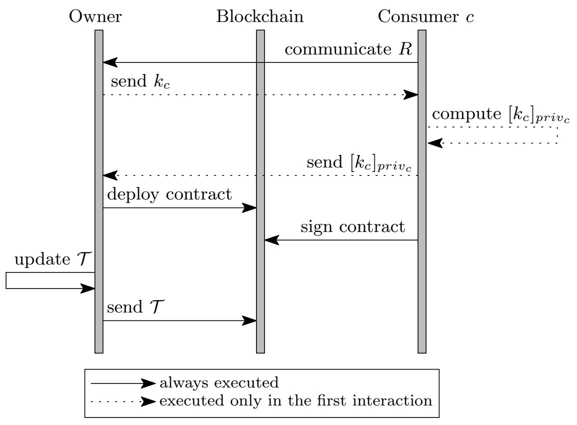


Fig. 9 Interaction protocol for token catalog update in [23]

A possible solution to such issues leverages blockchain and smart contracts only to finalize the monetary transaction and, at the same time, obtain a public and verifiable commitment of the owner to grant the paid access. The smart contract can then be arbitrarily formulated in such a way to securely keep track of the purchase, according to the following logic: "upon receiving `price` from c for the set R of resources, $\mathsf{cap}(c)$:=$\mathsf{cap}(c) \cup R$, and the token catalog is updated to grant c access to $\mathsf{cap}(c)$" [23], where `price` is the agreed price to be paid by consumer c to the owner to get access to R. The overall interplay between the owner and consumer c, including the deployment of the smart contract on the blockchain, is then regulated by the interaction protocol in Figure 9, which works as follows.

1. Consumer c communicates off-chain the set R of resources that she wants to purchase to the owner.
2. If necessary (i.e., in the first interaction, where c has not yet received the encryption key k_c from the owner), the owner generates k_c and sends it off-chain to c, who signs it with her own private key and sends back to the owner the signed key, denoted in Figure 9 as $[k_c]_{priv_c}$ (the signature is needed for the audit process described next).
3. The owner deploys the smart contract following the logic illustrated above.
4. Consumer c accesses, executes and signs the smart contract, triggering the money transfer to the owner.
5. The owner updates the key derivation structure as needed to grant c the agreed accesses.
6. The owner stores the updated token catalog on the blockchain.

It is interesting to note that such a slim interaction protocol prevents the possibility for a malicious consumer of not paying for a resource (the key derivation structure is updated after the execution of the smart contract) as well as for a malicious owner of claiming that she has not received the agreed money (thanks to the public nature of blockchains). Also, keys are safe since the derivation structure is updated locally by the owner. However, the interaction protocol cannot prevent malicious owners from refusing to update the derivation structure after receiving a payment, nor malicious consumers from claiming that access has not been provided despite the payment. Unfortunately, such misbehaviors cannot be prevented since the key derivation structure is updated locally. However, they can be easily detected and exposed, through a very simple *audit process* that can be executed whenever they are suspected. The audit protocol leverages the fact that the token catalog and the capability lists are stored on-chain (and hence are tamper-proof), and that the keys agreed with consumers are signed by them.

The audit process can be invoked by both owners and consumers, and simply requires a designated trusted auditor to explicitly check whether, starting from the key k_c of the involved consumer c, the set of tokens stored on-chain actually permits to correctly derive the keys k_r for all $r \in \mathsf{cap}(c)$. This is simply done by querying the token catalog and performing the key derivation that is allowed by tokens. If the entire derivation succeeds, then the owner has behaved correctly and hence the malicious behavior is at c's side, falsely claiming of not having being granted an access that actually is enabled. On the contrary, if the derivation fails, the owner has not done what she promised in the smart contracts, preventing a legitimate access. The availability of an audit process detecting and exposing not only misbehaviors, but also the misbehaving party, counteracts the possibility of misbehaviors and incentivizes all parties to behave correctly not to reduce their credibility and reputation that (like in any real-world transaction) are key factors for having subjects engaging in negotiations and transactions.

We close this section with some notes on the audit process. The reliability of its results is based on the correctness and freshness of the tokens and labels, of the capability lists, and of the starting key k_c. Correctness and freshness of tokens, labels, and capability lists are guaranteed by the fact that they are stored on-chain. For keys, these properties are guaranteed by the fact that they are signed by the respective owners (the owner cannot forge and c cannot repudiate her signature). This is the reason why keys are signed in the interaction protocol (and signed keys are of course used in the computation and update of tokens in the key derivation structure). It is also interesting to note that the combined adoption of on-chain storage, interaction protocol, and audit process permits to arbitrarily check the correctness of the derivation structure at any point in time, since all communications and relevant data (i.e., tokens and capability lists) are stored on-chain and hence tamper-proof.

5 Conclusions

We have addressed the problem of allowing data owners to share data with interested consumers in the context of digital data markets. First, we have investigated the specification and enforcement of requirements and preferences that can be used by owners to select the most suitable cloud plans for storing their data collections. Then, we have investigated the problem of controlled data sharing in the market, ensuring that owners remain in control of who accesses which portions of their data, and that they receive incentives for trading and sharing their data with others. For both issues, we have illustrated recent solutions that can be adopted. These are two key aspects, out of many, to ensure that owners remain in control in digital data markets. Other relevant aspects include, but are not limited to, the specification and enforcement of complex access and usage policies, possibly based on purpose and secondary use, and the enforcement of privacy-aware retrieval and analytics.

Acknowledgments

This work was supported in part by the EC within the H2020 Program under projects MOSAICrOWN and MARSAL, and by the Italian Ministry of Research within the PRIN program under project HOPE.

References

1. S. Zuboff, *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. New York: PublicAffairs, 2019.
2. S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati, “Towards owners’ control in digital data markets,” *IEEE Systems Journal (ISJ)*, 2020, (to appear).
3. R. Jhawar, V. Piuri, and M. Santambrogio, “Fault tolerance management in cloud computing: A system-level perspective,” *IEEE Systems Journal (ISJ)*, vol. 7, no. 2, pp. 288–297, 2013.
4. R. Jhawar and V. Piuri, “Fault tolerance and resilience in cloud computing environments,” in *Computer and Information Security Handbook, 2nd Edition*, J. Vacca, Ed. Morgan Kaufmann, 2013, pp. 125–142.
5. Y. Guo, Z. Mi, Y. Yang, H. Ma, and M. S. Obaidat, “Efficient network resource preallocation on demand in multitenant cloud systems,” *IEEE Systems Journal (ISJ)*, vol. 13, no. 4, pp. 4027–4038, 2019.
6. S. De Capitani di Vimercati, S. Foresti, G. Livraga, V. Piuri, and P. Samarati, “Supporting users in cloud plan selection,” in *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of his 70th Birthday*, P. Samarati, I. Ray, and I. Ray, Eds. Springer, 2018.
7. Y. Xie, Y. Guo, Z. Mi, Y. Yang, and M. S. Obaidat, “Loosely coupled cloud robotic framework for QoS-driven resource allocation-based Web service composition,” *IEEE Systems Journal (ISJ)*, vol. 14, no. 1, pp. 1245–1256, 2020.
8. A. Li, X. Yang, S. Kandula, and M. Zhang, “CloudCmp: Comparing public cloud providers,” in *Proc. of the 10th ACM Internet Measurement Conference (ACM IMC)*, Melbourne, Australia, November 2010.
9. S. K. Garg, S. Versteeg, and R. Buyya, “A framework for ranking of cloud computing services,” *Future Generation Computer Systems (FGCS)*, vol. 29, no. 4, pp. 1012–1023, 2013.
10. S. Ding, Z. Wang, D. Wu, and D. L. Olson, “Utilizing customer satisfaction in ranking prediction for personalized cloud service selection,” *Decision Support Systems*, vol. 93, pp. 1–10, 2017.
11. N. Ghosh, S. K. Ghosh, and S. K. Das, “SelCSP: A framework to facilitate selection of cloud service providers,” *IEEE Transactions on Computers (TCC)*, vol. 3, no. 1, pp. 66–79, 2015.

12. L. Qu, Y. Wang, M. A. Orgun, L. Liu, H. Liu, and A. Bouguettaya, “CCCloud: Context-aware and credible cloud service selection based on subjective assessment and objective assessment,” *IEEE Transactions on Services Computing (TSC)*, vol. 8, no. 3, pp. 369–383, 2015.
13. M. Tang, X. Dai, J. Liu, and J. Chen, “Towards a trust evaluation middleware for cloud service selection,” *Future Generation Computer Systems (FGCS)*, vol. 74, pp. 302–312, 2017.
14. Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, “QoS ranking prediction for cloud services,” *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 24, no. 6, pp. 1213–1222, 2013.
15. V. Casola, A. De Benedictis, M. Eraşcu, J. Modic, and M. Rak, “Automatically enforcing security SLAs in the cloud,” *IEEE Transactions on Services Computing (TSC)*, vol. 10, no. 5, pp. 741–755, 2017.
16. J. Luna, N. Suri, M. Iorga, and A. Karmel, “Leveraging the potential of cloud security service-level agreements through standards,” *IEEE Cloud Computing*, vol. 2, no. 3, pp. 32–40, 2015.
17. Cloud Security Alliance, “Cloud Control Matrix v3.0.1,” https://cloudsecurityalliance.org/research/ccm/.
18. R. Jhawar, V. Piuri, and P. Samarati, “Supporting security requirements for resource management in cloud computing,” in *Proc. of the 15th IEEE International Conference on Computational Science and Engineering (IEEE CSE)*, Paphos, Cyprus, December 2012.
19. S. De Capitani di Vimercati, S. Foresti, G. Livraga, V. Piuri, and P. Samarati, “Supporting user requirements and preferences in cloud plan selection,” *IEEE Transactions on Services Computing (TSC)*, November 2017, (to appear).
20. S. Sundareswaran, A. Squicciarini, and D. Lin, “A brokerage-based approach for cloud service selection,” in *Proc. of the 5th IEEE International Conference on Cloud Computing (IEEE CLOUD)*, Honolulu, HI, USA, June 2012.
21. S. De Capitani di Vimercati, S. Foresti, G. Livraga, V. Piuri, and P. Samarati, “A fuzzy-based brokering service for cloud plan selection,” *IEEE Systems Journal (ISJ)*, vol. 13, no. 4, pp. 4101–4109, 2019.
22. P. Samarati and S. De Capitani di Vimercati, “Cloud security: Issues and concerns,” in *Encyclopedia on Cloud Computing*, S. Murugesan and I. Bojanova, Eds. Wiley, 2016.
23. S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati, “Empowering owners with control in digital data markets,” in *Proc. of the 12th IEEE International Conference on Cloud Computing (IEEE CLOUD)*, Milan, Italy, July 2019.
24. M. Atallah, M. Blanton, N. Fazio, and K. Frikken, “Dynamic and efficient key management for access hierarchies,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 3, pp. 18:1–18:43, 2009.
25. S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, “Encryption policies for regulating access to outsourced data,” *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 2, pp. 12:1–12:46, 2010.
26. Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” in *Proc. of the 2017 IEEE International Conference on Big Data (IEEE BigData)*, Boston, MA, USA, December 2017.
27. R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, “Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts,” in *Proc. of the 4th IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*, Stockholm, Sweden, June 2019.