Security-Aware Data Allocation in Multicloud Scenarios

Sabrina De Capitani di Vimercati, *Senior Member, IEEE,* Sara Foresti, *Senior Member, IEEE,*
Giovanni Livraga, *Member, IEEE,* Vincenzo Piuri, *Fellow, IEEE,* and Pierangela Samarati, *Fellow, IEEE.*
E-mail: *firstname.lastname*@unimi.it

Abstract—When moving large and heterogeneous data collections to the cloud, a key requirement concerns the selection of the most suitable (set of) cloud service(s) for outsourcing. Not only can different resources have different characteristics and requirements, but different cloud providers can also offer different services and security guarantees, and can have different costs. Selecting a single service for outsourcing an entire data collection can result in a non-optimal solution, as a single service satisfying, at reasonable costs, all the requirements specified by the data owner might not exist. Selecting a set of services could instead ensure the satisfaction of the requirements, possibly with economic advantages. In this paper, we address this problem and present a flexible and expressive, yet simple model for supporting data owners in identifying a proper allocation of their resources to a set of cloud services. Our model allows data owners to specify in an easy and intuitive way protection requirements operating at the granularity level of single resource (or class thereof), and representing the minimum security guarantees that a cloud service must offer to store resources. Resources can be outsourced in plaintext or encrypted form, depending on their requirements and on what is the most convenient allocation. Data owners can then also specify global allocation requirements that apply to the overall allocation, to reduce the burden on their side and to avoid excessive fragmentation of the resource collection. We solve the problem of finding an allocation that satisfies both the protection and the global allocation requirements, while minimizing economic costs, by formulating it as a binary programming problem, thus allowing the use of existing techniques for its efficient solution.

Index Terms—Multicloud, Allocation, Protection requirements, Global allocation requirements

1 Introduction

Relying on cloud services for data storage and management has became a popular solution for almost any data owner, ranging from individual users storing their photos to external services for freeing space on their smartphones, to big organizations leveraging commercial cloud platforms to outsource their information systems. Since the cloud market offers a multitude of services characterized by different features, the first step when moving data to the cloud requires to select the right service(s). This task can be complex and entail some critical evaluations. First, it is necessary to understand the characteristics and requirements of the data (e.g., are they sensitive? are they subject to specific regulations and restrictions? how frequently and how do they need to be accessed?). It is then necessary to understand the characteristics of the available cloud services, to determine which are acceptable with respect to resource protection requirements. Also, since different cloud services come at different price points, a proper cost evaluation needs to be done.

The definition of requirements and their evaluation with respect to potential services may be complicated even for scenarios where small data collections with simple requirements are outsourced, and becomes more complex whenever the resources to be outsourced are multiple, critical, and with specific, contrasting, or non-comparable requirements. This can be the case, for example, of mid- or large-size organizations or public entities, wishing to move to the cloud their entire informative content, where data can be heterogeneous and range from sensitive data with stringent confidentiality requirements, to general information whose public accessibility should be guaranteed. In this case, each resource (or class of resources) can have peculiar needs, based on its nature: while sensitive data need services trusted for security/confidentiality, public data would benefit from a service offering high availability. In such a scenario, one possible solution would be the selection of a single cloud service for storing the entire data collection (e.g., [1], [2]). This strategy could produce solutions that either do not satisfy all requirements (as a single service perfectly satisfying all requirements of all resources might not exist) or are non-cost-effective (e.g., when selecting a costly service to satisfy the requirements of the most critical resources). A second option can be to rely on a set of providers/services for data storage to ensure that all requirements are satisfied and the cost is acceptable. This strategy, pursued in this paper, can accommodate fine-grained requirements of heterogeneous resource collections.

Finding a set of cloud services to store a heterogeneous data collection can be a difficult problem in itself, and entails a number of questions that need to be solved. For instance, a contract with a cloud service provider inevitably requires some management overhead, so relying on too many services might not be an effective strategy. Hence, how many services are to be selected? Are enough resources stored at each provider, so to compensate the burden of a contract management? Should resources expected to be frequently

- *Sabrina De Capitani di Vimercati, Sara Foresti, Giovanni Livraga, Vincenzo Piuri, and Pierangela Samarati are with the Computer Science Department, Università degli Studi di Milano, 20133 Milan – Italy.*

accessed together be managed by the same service? Should a set of resources not be jointly visible at the same service? These are examples of aspects that should be addressed when computing a multicloud allocation. In this paper, we address these problems and propose a model for a security-aware data allocation in multicloud scenarios. Our approach accounts for the possible application of owner-side encryption, so to take into account the possibility of relying on more affordable (but possibly not fully trusted) cloud services to store a (protected) portion of the resources.

The contribution of the paper is threefold. First, we provide an approach to express protection requirements that enable abstracting from low-level characteristics in their specification. Our separation of security properties from requirements nicely fits complex scenarios where protection requirements may need to consider directives and security conditions specified at a more general level (e.g., organization-wide). Second, we identify additional global allocation requirements enabling the specification and enforcement of restrictions on the overall resource allocation (in contrast to requirements on individual resources). Third, we provide a possible formulation of the problem of finding an allocation that satisfies protection and global allocation requirements while minimizing economic costs of external data management.

The remainder of this paper is organized as follows. Section 2 illustrates our reference scenario and introduces the notion of resource allocation to services taking into account owner-side encryption. Section 3 presents our security model, based on abstract security properties and their mapping to parameters and metadata of services. Sections 4 and 5 illustrate how such security model is used for classifying available cloud services and for formulating resource protection requirements. Section 6 introduces global allocation requirements. Section 7 illustrates when an allocation of resources (characterized by protection and global allocation requirements) to services (classified according to our security model) can be considered correct with respect to all specified requirements, and minimal with respect to economic costs. Section 8 introduces a formulation of the problem of computing a correct and minimal allocation as a binary programming problem, easily solvable using existing techniques. Section 9 discusses the possible advantages of owner-side encryption. Section 10 discusses related work. Finally, Section 11 concludes the paper.

2 Reference scenario

We consider a reference scenario characterized by cloud providers offering cloud service plans (*cloud services*, for short) and *data owners* wishing to outsource the management of their resources to external cloud providers. The goal of this paper is to support data owners in allocating resources to cloud services that better match their requirements. Such requirements represent the security needs of the resources (e.g., sensitive resources should be managed by cloud services offering "strong" security guarantees) as well as constraints on the overall allocation of resources to cloud services (e.g., some resources often accessed together should be managed by the same cloud service). Our solution allows data owners to express their requirements in an easy way by referring to high-level *properties*, in contrast to low-level *configuration parameters* describing the characteristics of cloud services. In the following, we model the resources to be outsourced as a set $\mathcal{R}$ and the cloud services offered by cloud providers as a set $\mathcal{S}$. Note that our model is agnostic with respect to the granularity of resources, which may be single data items or (as expected) collections thereof. In the following, we simply use the term resource in a generic sense, to refer to either individual resources or classes thereof depending on the specific application scenario.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
`provider`	Ghost	Cloudy	Amaron	Mist	Mhard	GoGo	NewCloud
`loc`	EU	US	EU	US	EU	US	EU
`cert`	certA	certA		certC	certA		certB
`audit`			auditA				auditB
`uptime`	99.999	99.998	99.970	99.980		99.997	99.960

Figure 1. An example of cloud services

Cloud services are characterized by a set $\mathcal{A}$ of *attributes*. In our work, we do not restrict our approach to any specific predefined set of attributes and assume to refer to a generic set of attributes encompassing all possible properties to be considered, including configuration parameters (e.g., uptime) and metadata associated with cloud services (e.g., name of the cloud provider). We assume names and values of the attributes to be taken from a known ontology/vocabulary [1]. A cloud service can then be seen as a tuple containing the values for the applicable attributes in $\mathcal{A}$. We use notation $s[a]$, with $a \in \mathcal{A}$, to denote the value of attribute a for cloud service $s \in \mathcal{S}$. Figure 1 illustrates the cloud services of our running example, considering set $\mathcal{A}$ of attributes: `provider`, the name of the provider offering the cloud service; `loc`, the geographical location of the storage servers used by the provider; `cert`, the security certification for the service; `audit`, the authority in charge of the security auditing; and `uptime`, the service uptime guaranteed in the Service Level Agreement. Note that, in the figure, empty cells denote the fact that, for a cloud service, the value of the corresponding attribute is unknown or not defined. For instance, the value of attribute `audit` is undefined for s_1.

Accounting for owner-side encryption, as common in many emerging scenarios, we consider that resources can be outsourced in plaintext or in encrypted form. The advantage of outsourcing resources in encrypted form is that they are also protected from the service storing them and therefore, in principle, they can be managed by potentially less trusted, but more affordable, providers. The allocation of a resource to a cloud service must then also specify whether the resource is outsourced in plaintext or encrypted. Allocation is formally defined as follows.

Definition 2.1 (Allocation). Given a set $\mathcal{S}$ of cloud services and a set $\mathcal{R}$ of resources, an *allocation* for $\mathcal{R}$ over $\mathcal{S}$ is a function $\alpha{:}\mathcal{R} \rightarrow \mathcal{S} \times \{\circ, \bullet\}$ that associates with each resource $r \in \mathcal{R}$ a pair $\langle s, m \rangle$, with $m \in \{\circ, \bullet\}$.

Given a resource r, $\alpha(r) = \langle s, m \rangle$ states that resource r is allocated to cloud service s and that r is visible ($m = \circ$), meaning that the resource is outsourced in plaintext form, or encrypted ($m = \bullet$), meaning that the resource is encrypted at the owner side before outsourcing. We denote the allocation mode (plaintext or encrypted, respectively) with a (empty or full, respectively) circle to connect formal notation to our graphical representation that will use gray boxes to denote application of encryption.

3 Abstract security properties

We now introduce our security model, based on abstract security properties, which will be used to classify cloud services (Section 4) as well as for formulating resource protection requirements (Section 5).

Abstract security properties are high-level concepts associated with a domain of labels each of which corresponds to the satisfaction of given formulas over the attributes of the cloud services. There are two main advantages in using abstract security properties. First, the data owner can reason about the needed requirements without worrying about the specific attributes of the cloud services that contribute to obtain the desired need/protection. Second, the mapping between the requirements and the attributes is made only once, thus facilitating the management of requirements that will just need to refer to high-level concepts. This separation between abstract properties and attributes results even more advantageous in scenarios of cloud brokering services or of large organizations where one entity (the broker or the company) can define the abstract properties and their correspondence in terms of attributes, and resource owners/administrators can then specify requirements on resources with reference to such abstract properties.

Natural abstract security properties are the classical CIA: Confidentiality, Integrity, Availability, but others can also be added depending on the richness (or granularity) desired. Our model considers a generic set $\mathcal{P}$ of properties defined by the data owner/broker. Values associated with properties are high-level labels that characterize a level of satisfaction with respect to the property. We assume labels to be ordered, with higher labels representing better satisfaction of the property (intuitively corresponding to stronger conditions over the low-level attributes of cloud services).

To maintain examples easy to follow and draw, in the paper, we refer to a set $\mathcal{P}=\{\mathbf{C},\mathbf{A}\}$ of two properties ($\mathbf{C}$onfidentiality and $\mathbf{A}$vailability) each with three security labels corresponding to High (HC and HA), Medium (MC and MA), and Low (LC and LA) values. A default bottom label $\perp$ is assumed for all properties, denoting the label to be considered when the property is of no interest.

The correspondence between property labels and attributes of the cloud services is defined by rules associated with property labels. More precisely, each property $p \in \mathcal{P}$ is defined as a triple comprising: the labels associated with the property, the correspondence between the labels and values of attributes of cloud services, and a total order relationship among labels, as formalized by the following definition.

Definition 3.1 (Security properties). Given a set $\mathcal{A}$ of attributes, a set $\mathcal{P}$ of *security properties* over it is a set where each property $p \in \mathcal{P}$ is a triple $\langle \mathcal{L}^p, e^p, \succ^p \rangle$ with:

- $\mathcal{L}^p$ a set $\{l_1, \ldots, l_m\}$ of labels;
- e^p:$\mathcal{L}^p \rightarrow E$ a function that associates with each label $l_i \in \mathcal{L}^p$ an expression $e^p(l_i) \in E$ over attributes in $\mathcal{A}$;
- $\succ^p$ a total order relationship over $\mathcal{L}^p$.

Property p	**Label** l	**Expression** $e^p(l)$
$\mathbf{C}$	HC	loc=EU ∧ cert=certA
	MC	cert=certA ∨ audit=auditA
	LC	cert=certB ∨ audit=auditB
	$\perp$	true
$\mathbf{A}$	HA	uptime>99.99%
	MA	uptime>99.95%
	LA	uptime>99.90%
	$\perp$	true

Figure 2. Security properties of the running example

We do not impose any restriction on the expressions associated with labels via function e^p. Our model is therefore agnostic with respect to this and can accommodate any function over the attributes, from basic conditions to more expressive languages (e.g., [1]). This being said, for concreteness and without loss of generality, in this paper we consider expressions to simply be Boolean formulas combining conditions over the values for the attributes characterizing services. In the following, given two labels $l_i, l_j \in \mathcal{L}^p$ we write $l_i \succeq^p l_j$ iff $l_i \succ^p l_j$ or $l_i = l_j$.

Figure 2 illustrates the properties ($\mathcal{P}=\{\mathbf{C}, \mathbf{A}\}$) and labels ($\mathcal{L}^{\mathbf{C}}$={HC, MC, LC, $\perp$} with HC$\succ^{\mathbf{C}}$MC$\succ^{\mathbf{C}}$LC$\succ^{\mathbf{C}}\perp$; $\mathcal{L}^{\mathbf{A}}$={HA, MA, LA, $\perp$} with HA$\succ^{\mathbf{A}}$MA$\succ^{\mathbf{A}}$LA$\succ^{\mathbf{A}}\perp$) of our running example together with the expressions associated with the labels. For instance, confidentiality label HC requires location of the service to be EU and certification to be certA, while confidentiality label LC requires either a certification certB or audit authority auditB. Note that the expressions associated with the security labels of the different properties are assumed to be independent, meaning that the satisfaction of the expression associated with label l_i does not necessarily imply the satisfaction of the expression associated with label l_j, with $l_i \succ^p l_j$. This is particularly evident, for instance, whenever conditions associated with labels are mutually exclusive. For instance, if audit authority auditA is to be considered more trusted than authority auditB, the confidentiality label associated with expression 'audit=auditA,' will clearly be higher than the one associated with expression 'audit=auditB'. At the same time, however, satisfying the first condition clearly impedes the satisfaction of the second condition (more details will be given in Section 4).

The consideration of different properties (e.g., Confidentiality and Availability in our running example) implies the consideration of a tuple $[l_{i_1}, \ldots, l_{i_n}]$ of labels (with a label for each property), which we call *security class* and denote as c. For instance, [HC,HA] is a security class for our running example. The total order relationship holding within the labels of the different properties induces then a partial order relationship on the security classes forming a lattice as formalized by the following definition.

Definition 3.2 (Security lattice). Given a set $\mathcal{P}=\{p_1, \ldots, p_n\}$ of security properties, a *security lattice* $\mathcal{H}$ over $\mathcal{P}$ is a pair $\langle \mathcal{C}, \succeq \rangle$ where $\mathcal{C}=\mathcal{L}^{p_1} \times \ldots \times \mathcal{L}^{p_n}$ and $\succeq$ is the dominance relationship over $\mathcal{C}$ such that $\forall c_i, c_j \in \mathcal{C}$, c_i dominates c_j, denoted $c_i \succeq c_j$, iff $c_i[k] \succeq^{p_k} c_j[k]$, k=1, …, n.

In other words, a security class dominates another one iff the dominance relationship holds for each of its components (i.e., labels in the tuple). For instance, [HC, HA]$\succeq$[MC, MA]. Figure 3 illustrates the lattice for the properties of our

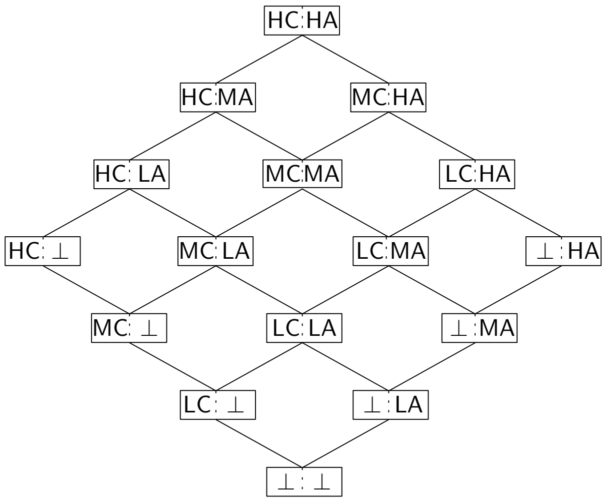


Figure 3. Security lattice over the security properties in Figure 2

running example. The expression associated with a security class (i.e., the conditions corresponding to it) is the conjunction of the expressions associated with its components. Formally, given a security class $c \in \mathcal{C}$, $e(c) = \bigwedge_{j=1}^{n} e^{p_j}(c[j])$. For instance, with reference to Figure 2, $e([\mathsf{HC}, \mathsf{HA}])$ = (`loc`=EU ∧ `cert`=certA) ∧ (`uptime`>99.99%).

4 Cloud service classification

The consideration of abstract security properties (to which resource protection requirements will refer) allows data owners to avoid worrying about the attributes characterizing cloud services. Before illustrating how resource requirements can be specified and enforced for selecting suitable cloud services, we illustrate how services' characteristics map onto property labels and therefore onto security classes.

The expression associated with each property label dictates the conditions that should be satisfied on the attributes of cloud services for the label to apply. In other words, the conditions that a cloud service should satisfy for being considered suitable for resources having that label. This introduces a natural mapping between services (their low-level attributes) and property labels composing security classes.

We say that a cloud service s satisfies a security label $l \in \mathcal{L}^p$, denoted $s \models l$, if the values of the attributes in s satisfy expression $e^p(l)$, that is, they make it evaluate to true. Similarly, a cloud service satisfies a security class c, denoted $s \models c$, if the values of its attributes satisfy $e(c)$. For instance, with reference to our running example (services in Figure 1 and labels in Figure 2) $s_1 \models [\mathsf{HC}, \mathsf{HA}]$, $s_1 \models [\mathsf{HC}, \mathsf{MA}]$, $s_1 \models [\mathsf{HC}, \mathsf{LA}]$, and $s_1 \models [\mathsf{HC}, \perp]$. Clearly, any service satisfies $\perp$.

Note that, since we do not assume any implication among expressions associated with labels, satisfaction of a security class for a given service is not necessarily monotone. Hence, a service s might satisfy a security class c_x but might not satisfy a class c_y dominated by it, meaning that the values of the attributes in s satisfy $e(c_x)$ but not $e(c_y)$. For instance, $s_1 \models [\mathsf{HC}, \mathsf{HA}]$, but $s_1 \not\models [\mathsf{LC}, \mathsf{HA}]$, because $s_1 \models \mathsf{HC}$ but $s_1 \not\models \mathsf{LC}$. This non-monotonicity is completely in line with the freedom in the specification of expressions and the fact that, as already noted, expressions might be mutually exclusive. This being said, given the semantics of labels, it is clear that if a service is suitable for a given security class it is also suitable for lower (dominated) security classes. Monotonicity in terms of satisfaction of the corresponding expression can be simply ensured by extending the expression associated with lower classes with the disjunction of the expressions associated with classes that dominate them. We are therefore interested in the highest security class that a cloud service satisfies, which we refer to as *security class of the service*, formalized as follows.

Definition 4.1 (Security class of a service). Given a cloud service $s \in \mathcal{S}$ and a security lattice $\mathcal{H}$=$\langle \mathcal{C}, \succeq \rangle$ over $\mathcal{P}$, the *security class* of s, denoted $\lambda(s)$, is the security class $c_i \in \mathcal{C}$ such that $s \models c_i$, and $\nexists c_j \in \mathcal{C}$ with $c_j \succeq c_i$ and $s \models c_j$.

Intuitively, the security class of a cloud service represents the maximal security guarantee that it offers. For instance, the security class of s_1 is [HC,HA]. It is interesting to note that the security class of a service is unique, as stated by the following theorem.

Theorem 4.1. Given a cloud service $s \in \mathcal{S}$ and a security lattice $\mathcal{H}$=$\langle \mathcal{C}, \succeq \rangle$ over $\mathcal{P}$, $\exists! c \in \mathcal{C} : c = \lambda(s)$.

PROOF: Consider an arbitrary service s. To prove the theorem, we show that: 1) the security class of s exists; and 2) the security class of s is unique.

1) Proof of the existence of the security class of s is trivial, since the lattice includes class $c_\perp$=$[\perp, \ldots, \perp]$ with value $\perp$ (with $e^p(\perp)$=`true`) for each property. Since, by definition, $s \models \perp$, then $s \models c_\perp$.
2) We prove that the security class of s is unique by contradiction. Suppose that $\exists c_{i_1}, c_{i_2}$ such that $\lambda(s)$=c_{i_1} and $\lambda(s)$=c_{i_2}. By Definition 4.1, we have that $c_{i_1} \not\succeq c_{i_2}$, and $c_{i_2} \not\succeq c_{i_1}$. Since $\lambda(s)$=c_{i_1}, then $s \models c_{i_1}$, meaning that $\forall p_k \in \mathcal{P}$, $s \models c_{i_1}[k]$. Similarly, we have that $\forall p_k \in \mathcal{P}$, $s \models c_{i_2}[k]$. Consider now a security class c_j such that $\forall p_k \in \mathcal{P} : c_j[k]$=max$(c_{i_1}[k], c_{i_2}[k])$. Since the lattice is complete, c_j belongs to the lattice. Also, since labels are totally ordered, we have that $c_j \succeq c_{i_1}$ and $c_j \succeq c_{i_2}$. Also, since $s \models c_{i_1}$ and $s \models c_{i_2}$, we have that $s \models c_j$, contradicting the hypothesis that $\lambda(s)$=c_{i_1} and $\lambda(s)$=c_{i_2}. □

Given the non-monotonicity with respect to the satisfaction of the conditions along the lattice, the determination of the highest security class would in principle require to walk down the lattice, evaluating conditions from the top to the bottom (considering then the highest security class for which the condition is satisfied). Since the expression associated with a security class is simply the conjunction of the expressions of the class components, there is no need to walk down the lattice: the evaluation can be done separately on each component, retrieving the highest label satisfied for each property, as illustrated by the simple algorithm in Figure 4. The class of the service will then be the class represented by the tuple of the retrieved labels. The security classes for the services of our running example are: $\lambda(s_1) = [\mathsf{HC}, \mathsf{HA}]$, $\lambda(s_2) = [\mathsf{MC}, \mathsf{HA}]$, $\lambda(s_3) = [\mathsf{MC}, \mathsf{MA}]$, $\lambda(s_4) = [\perp, \mathsf{MA}]$, $\lambda(s_5) = [\mathsf{HC}, \perp]$, $\lambda(s_6) = [\perp, \mathsf{HA}]$, and $\lambda(s_7) = [\mathsf{LC}, \mathsf{MA}]$.

Algorithm 1 **(Security class of cloud service s).**

```
for each $p \in \mathcal{P}$ do
    $\mathcal{L}^{p\prime}$ := $\mathcal{L}^{p}$ /* create a copy of the set $\mathcal{L}^{p}$ of labels */
    $l$ := Top($\mathcal{L}^{p\prime}$) /* highest label in $\mathcal{L}^{p\prime}$*/
    $c[p]$ := null /* label associated with $s$ for property $p$ */
    repeat
        if $s \models l$ /* verify if $s$ satisfies security label $l$ */
            then $c[p]$ := $l$ /* $l$ is the label associated with $s$ for $p$ */
            else $\mathcal{L}^{p\prime}$ := $\mathcal{L}^{p\prime} \setminus \{l\}$ /* otherwise, move to the next label */
                $l$ := Top($\mathcal{L}^{p\prime}$)
    until $c[p] \neq$ null
$\lambda(s)$ := $c$ /* security class of $s$ */
```

Figure 4. Algorithm for computing the security class of a cloud service

5 Resource protection requirements

We now illustrate the specification of resource protection requirements. Accommodating for the possibility of owner-side encryption of resources, our model also takes into account that requirements on the same resource might be different depending on whether the resource is outsourced in plaintext or encrypted. For instance, a resource with HC as confidentiality requirement might downgrade the requirement to LC in its encrypted form. Intuitively, the security class associated with a (plaintext/encrypted) resource dictates the policy restrictions that the service should satisfy for storing the (plaintext/encrypted) resource.

Resource protection requirements are then specified associating with each resource r (or class thereof, as best suited) two security classes, corresponding to the requirements for the plaintext and encrypted resource, respectively, as formalized by the following definition.

Definition 5.1 (Resource protection requirements). Given a set $\mathcal{R}$ of resources and a security lattice $\mathcal{H} = \langle \mathcal{C}, \succeq \rangle$, the *resource protection requirements* for $\mathcal{R}$ are a pair of functions $\langle \lambda^\circ, \lambda^\bullet \rangle$, with $\lambda^\circ : \mathcal{R} \to \mathcal{C} \cup \{-\}$, and $\lambda^\bullet : \mathcal{R} \to \mathcal{C} \cup \{-\}$.

According to this definition, the resource protection requirements for a set $\mathcal{R}$ of resources are seen as two functions λ° and $\lambda^\bullet$, each associating with each resource a security class. To express cases in which the resource cannot be outsourced (in its plaintext or encrypted form), we extend λ° and $\lambda^\bullet$ to include special value $-$, which indicates that a resource cannot be outsourced in a given form. More precisely: if $\lambda^\circ(r)$=$-$, the resource cannot be outsourced in plaintext; it must be encrypted by the data owner before being outsourced to the cloud. Analogously, if $\lambda^\bullet(r)$=$-$, resource r must be outsourced in plaintext; it cannot be outsourced in its encrypted form. The case $\lambda^\circ(r)$=$-$ can accommodate for resources particularly sensitive for which the data owner (for company policy or due to legislation restrictions) wishes to impose external storage only in encrypted form. The case $\lambda^\bullet(r)$=$-$ can accommodate for resources whose availability in the clear is needed for access (as encryption would impede functionality needed over them).

In our running example, we consider the following five (classes of) resources, summarized in Figure 5 together with their protection requirements.

	λ°	$\lambda^\bullet$
`admin`	[MC, HA]	[LC, HA]
`agreements`	[LC, MA]	[⊥, MA]
`suppliers`	[MC, MA]	–
`projects`	[HC, HA]	–
`archive`	–	[⊥, LA]

Figure 5. Resource protection requirements for the resources of the running example

- `admin` (administrative files) can be stored in either plaintext or encrypted form and requires High Availability for both, with Medium Confidentiality if in plaintext and Low Confidentiality if encrypted;
- `agreements` (all files related to contracts with other parties) can be stored either in plaintext or encrypted form and requires Medium Availability for both, with Low Confidentiality if in plaintext and no confidentiality requirement if encrypted;
- `suppliers` (all files related to suppliers) can be stored in plaintext only and requires Medium Confidentiality and Medium Availability;
- `projects` (current projects): can be stored in plaintext only and requires High Confidentiality and High Availability;
- `archive` (old resources that must be kept for archival) can be stored only in encrypted form, requires Low Availability and does not have requirements on confidentiality.

The requirements specify, for each resource, the security classes that apply to it, with respect to both the original plaintext resource as well as its (owner-side) encrypted version. Such security classes are to be intended as the minimum guarantees to be provided for the resource for the different properties. In other words, the resource cannot be outsourced to a service that does not provide at least such guarantees. In line with the semantics associated with security classes and their lattice, services providing better security guarantees are instead suitable. Formally stated, this translates to demanding that a resource can be outsourced only to a service whose security class dominates the class of the resource. We refer to an allocation obeying such a restriction (i.e., satisfying the resource protection requirements) as *safe*, as formalized by the following definition.

Definition 5.2 (Safe allocation). Given a set $\mathcal{S}$ of cloud services, a set $\mathcal{R}$ of resources, and the resource protection requirements $\langle \lambda^\circ, \lambda^\bullet \rangle$ for $\mathcal{R}$, an allocation α for $\mathcal{R}$ over $\mathcal{S}$ is *safe* iff $\forall r \in \mathcal{R} : \alpha(r) = \langle s, m \rangle \Longrightarrow \lambda(s) \succeq \lambda^m(r)$.

Figure 6 graphically plots the classes of services and resources on the lattice of our running example. Each (plaintext or encrypted) resource appears on the left of the vertex corresponding to the class of its protection requirements. Plaintext resources appear in a vertex along one of the paths from the vertex where the encrypted resource appears to the top of the lattice. Each service appears on the right of the vertex corresponding to its security class. Graphically, a safe allocation allocates each resource (plaintext or encrypted, respectively) to a service appearing in a vertex along the path from the vertex where the (plaintext or encrypted, respectively) resource appears to the top of the lattice. Hence, different safe allocations can allocate a resource to different services. For instance, `agreements` can be allocated to s_7, s_3, s_2 or s_1 in plaintext and to s_4, s_6, s_7, s_3, s_2, and s_1 in

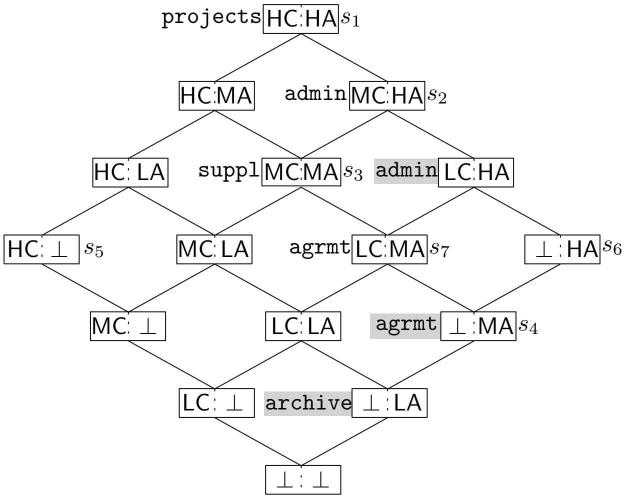


Figure 6. Security lattice of the running example with resources and cloud services

encrypted form. Figure 7 summarizes to which services each resource can be safely allocated. Cell $[r, s]$ has value: ○, if r can be allocated to s in plaintext only; ●, if r can be allocated to s in encrypted form only; ○ ●, if r can be allocated to s in plaintext or encrypted form; empty, if r cannot be safely allocated to s.

Allocation α_1 in Figure 8 is an example of a safe allocation for our running example. Allocation α_3 is instead not safe: suppliers is stored in encrypted form (violating $\lambda^\bullet$(suppliers)$=-$), archive is stored in plaintext (violating λ°(archive)$=-$), and projects is allocated plaintext to s_2 whose security class does not dominate λ°(projects).

As noted in the discussion above, given a set of services and resources with their corresponding classes, there might exist multiple safe allocations, differing in the service chosen for storage and/or the storing mode (i.e., plaintext or encrypted) of (some of) the resources. In addition to protection requirements on individual resources, we then allow data owners to express other requirements considering the overall allocation, that is, involving the complete set of resources, as we illustrate in the following section.

6 Global allocation requirements

Global allocation requirements impose conditions on the allocation of resources that do not refer to individual (classes of) resources but to sets of them or to the allocation overall. We identify three kinds of global requirements. The first two impose conditions on allocation of resources requesting their joint assignment to the same service or impeding their visibility to the same service. The third one imposes instead limitations on the involvement of multiple services. We define each of them in the following.

The first kind of requirement, called *co-location*, is formally defined as follows.

Definition 6.1 (Co-location). Given a set $\mathcal{R}$ of resources, a set CoL of *co-location requirements* over $\mathcal{R}$ is a set $\{loc_1, \ldots, loc_m\}$ such that $loc_i \subseteq \mathcal{R}, i = 1, \ldots, m$.

The semantics of a co-location requirement loc is that the resources in loc should be managed by the same service.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
admin	○ ●	○ ●					
agreements	○ ●	○ ●	○ ●	●		●	○ ●
suppliers	○	○	○				
projects	○						
archive	●	●	●	●		●	●

Figure 7. Possible safe plaintext (○) and encrypted (●) allocations of resources to services

	$\boldsymbol{\alpha_1}$	$\boldsymbol{\alpha_2}$	$\boldsymbol{\alpha_3}$	$\boldsymbol{\alpha_4}$
admin	s_1, ○	s_1, ●	s_2, ●	s_2, ○
agreements	s_3, ●	s_3, ●	s_3, ○	s_2, ○
suppliers	s_3, ○	s_3, ○	s_3, ●	s_2, ○
projects	s_1, ○	s_1, ○	s_2, ○	s_1, ○
archive	s_3, ●	s_3, ●	s_3, ○	s_7, ●

Figure 8. An example of allocations for our running example (α_1,α_2: safe and compliant; α_3: not safe but compliant; α_4: safe but not compliant)

The motivation for a co-location requirement might be that resources need to be accessible together, or be frequently accessed together, and therefore it is convenient to allocate them to the same cloud service. For instance, considering the set of resources in Figure 5, co-location requirement {admin,projects} in Figure 9 requires resources admin and projects to be allocated to the same cloud service. Note that a co-location requirement only requires the resources to be allocated to the same service, without imposing restrictions on the format (plaintext or encrypted) in which they are stored. The case where resources should be all allocated to the same service and be so in plaintext (similar to visibility requirements in [3]) is simply captured by setting, in addition to the co-location requirement, their protection requirement $\lambda^\bullet$ to value $-$.

Besides requesting some resources to be allocated to the same service, a data owner may also request resources not to be jointly visible at the same service. This is captured by the *separation* requirement, defined as follows.

Definition 6.2 (Separation). Given a set $\mathcal{R}$ of resources, a set Sep of *separation requirements* over $\mathcal{R}$ is a set $\{sep_1, \ldots, sep_n\}$ such that $sep_i \subseteq \mathcal{R}, i = 1, \ldots, n$.

The semantics of a separation requirement is to impede joint visibility over a collection of resources to a single service. Intuitively, this requirement captures confidentiality constraints (e.g., [4]) requesting no party to have joint visibility of a complete set of resources. Here, visibility means visibility on the plaintext content of the resources. For instance, considering our running example, separation requirement {agreements,suppliers} in Figure 9 states that resources agreements and suppliers should not be visible to the same cloud service. Similarly to what done for confidentiality constraints in [4], we assume a separation requirement to be satisfied if either at least one resource is allocated to a different service, or if at least one of them is encrypted. Again, as noted above, specific interpretation and corresponding representation mode can be regulated by properly setting the definition of the protection requirements $\langle\lambda^\circ, \lambda^\bullet\rangle$.

As noted in the previous section, several safe allocations may exist, all granting protection to resources and different alternatives may remain, all satisfying co-location

CoL ={{admin,projects}}
Sep ={{agreements,suppliers}}
Usage=[3, 30GB]

Figure 9. Global allocation requirements

and separation requirements. Such alternatives may differ with respect to the number and the use of each of the services. The last kind of requirement we consider aims at imposing some regulation on the involvement of different services. In fact, the use of multiple cloud services has also the disadvantage of introducing an overhead due to the spreading of resources. Also, the use of a cloud service for managing resources can be considered convenient when a considerable amount of resources is allocated to the cloud service itself. In fact, the use of a cloud service implies the signature of a contract between the involved parties with a consequent overhead for the data owner. Our last kind of requirement enables data owners to limit the overhead given by employing too many services, guaranteeing that the establishment of a contract is worth its management, and avoids excessive fragmentation of resources in the cloud. We define such requirement, called *service usage*, as follows.

Definition 6.3 (Service usage). A *service usage requirement* is a pair of integers Usage=$\langle$*max_services*,*min_storage*$\rangle$.

A service usage requirement $\langle$*max_services*,*min_storage*$\rangle$ imposes the allocation to use at most *max_services* services, employing each of them for at least *min_storage* storage occupation. For instance, with reference to our running example, service usage requirement [3, 30GB] in Figure 9 states that an allocation should use at most three services employing each of them for at least 30GB. Note that, for simplicity, we suppose that the same threshold *min_storage* applies to all cloud services. However, our model can be easily extended to adopt different thresholds for different cloud services. To define a service usage requirement, it is necessary to know the size of the resources in $\mathcal{R}$. Indeed, the size of a resource can increase due to encryption. While noting that the size increase depends on the adopted encryption scheme, we assume the scheme to be known when computing an allocation, and hence the ability to estimate the size of the encrypted version of resources.

A triple including the three requirements discussed above (namely, co-location, separation, and service usage) form the so-called *global allocation requirements*, formally defined as follows.

Definition 6.4 (Global allocation requirements). Given a set $\mathcal{R}$ of resources, the *global allocation requirements* for $\mathcal{R}$ are a triple $\langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle$ defined over $\mathcal{R}$.

Given a set $\mathcal{R}$ of resources and a set $\mathcal{S}$ of services, a *compliant allocation* for $\mathcal{R}$ over $\mathcal{S}$ is an allocation that satisfies the global allocation requirements for $\mathcal{R}$, as stated by the following definition. In the definition and in the reminder of the paper, we denote with $\alpha(r)[s]$ and $\alpha(r)[m]$ the service s and the plaintext/encrypted form m in which r is allocated by α.

Definition 6.5 (Compliant allocation function). Given a set $\mathcal{R}$ of resources, a set $\mathcal{S}$ of cloud services, and the global allocation requirements $\langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle$ for $\mathcal{R}$, an allocation α for $\mathcal{R}$ over $\mathcal{S}$ is said to be *compliant* iff:

	Size (GB)		Access frequency
	○	●	
admin	25	27.5	0.15
agreements	15	16.5	0.15
suppliers	20	22	0.15
projects	10	11	0.5
archive	100	110	0.05

Figure 10. Profile of the resources of the running example

1) $\forall loc \in \mathsf{CoL}, \forall r_i, r_j \in loc : \alpha(r_i)[s] = \alpha(r_j)[s]$;
2) $\forall sep \in \mathsf{Sep}, (\exists r_i, r_j \in sep : \alpha(r_i)[s] \neq \alpha(r_j)[s]) \vee (\exists r_i \in sep, \alpha(r_i)[m] = \bullet)$;
3) $\mid \bigcup_{r \in \mathcal{R}} \alpha(r)[s] \mid \leq$ *max_service*;
4) $\forall s_i \in \bigcup_{r \in \mathcal{R}} \alpha(r)[s]$: *storage*($s_i$)$\geq$ *min_storage*, with *storage*(s_i)= $\sum_{r \in \mathcal{R}:\alpha(r)[s]=s_i} size^{\alpha(r)[m]}(r)$.

An allocation is then compliant iff: *1)* for each co-location requirement, all resources appearing in it are allocated to the same service; *2)* for each separation requirement, at least one of the resources is allocated to a different service or is encrypted; *3)* no more than *max_service* services are involved in the allocation of resources; and *4)* storage used at each service employed in the allocation is at least *min_storage*. Allocations α_1, α_2, and α_3 in Figure 8 are compliant with respect to the global allocation requirements in Figure 9, considering the size of resources in Figure 10. It is easy to see that CoL is satisfied since admin and projects are both allocated to the same service (s_1 in α_1 and α_2, s_2 in α_3); Sep is satisfied since agreements (α_1 and α_2) or suppliers (α_3) is stored in encrypted form and hence is not jointly visible in plaintext at s_3; Usage is satisfied since the allocations employs only 2 services, and each service stores at least 30GB of resources. Allocation α_4 in Figure 8 is instead not compliant: admin and projects are not allocated to the same service (violating CoL); agreements and suppliers are both allocated plaintext to s_2 (violating Sep); and s_1 only stores 10GB (violating Usage).

7 Correct and minimal allocation

Protection requirements (Section 5) and global allocation requirements (Section 6) form the basis on which an allocation is to be computed, as we now illustrate (Section 7.1). We then introduce the notion of cost of an allocation, to determine the allocations that are more economically convenient (Section 7.2).

7.1 Correct allocation

Summarizing the different requirements introduced, we define an *allocation policy*, governing the allocation of resources to services, as comprising of resource protection and global allocation requirements, as captured by the following definition.

Definition 7.1 (Allocation policy). Given a set $\mathcal{R}$ of resources and a security lattice $\mathcal{H}$ over a set $\mathcal{P}$ of properties, an *allocation policy* for $\mathcal{R}$ is a pair $\langle\langle \lambda^{\circ}, \lambda^{\bullet}\rangle, \langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle\rangle$, with $\langle \lambda^{\circ}, \lambda^{\bullet}\rangle$ the protection requirements for $\mathcal{R}$ (Definition 5.1), and $\langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle$ the global allocation requirements for $\mathcal{R}$ (Definition 6.4).

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
st_cost(s)	50	50	40	50	10	80	50
tr_cost(s)	50	40	30	40	10	100	40

Figure 11. Storage and data transfer costs (USD¢/GB) for the cloud services of our running example

An allocation α is said to be *correct* with respect to an allocation policy iff it satisfies both the resource protection and the global allocation requirements identified in the policy, as formalized by the following definition.

Definition 7.2 (Correct allocation function). Given a set $\mathcal{R}$ of resources, a set $\mathcal{S}$ of cloud services, and a security lattice $\mathcal{H}$ over a set $\mathcal{P}$ of properties, an allocation α: $\mathcal{R} \rightarrow \mathcal{S} \times \{\circ, \bullet\}$ is said to be *correct* with respect to an allocation policy $\langle\langle\lambda^{\circ}, \lambda^{\bullet}\rangle, \langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle\rangle$ for $\mathcal{R}$ iff:

1) α is safe (Definition 5.2) and
2) α is compliant (Definition 6.5).

The definition above demands that a correct allocation for a set $\mathcal{R}$ of resources over a set $\mathcal{S}$ of services: 1) satisfies the protection requirements $\langle\lambda^{\circ}, \lambda^{\bullet}\rangle$ for $\mathcal{R}$; and 2) satisfies the global allocation requirements $\langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle$ for $\mathcal{R}$. Allocations α_1 and α_2 in Figure 8 are correct allocations for the resources in Figure 5 over the services in Figure 1. Allocations α_3 and α_4 are instead not correct, being α_3 not safe (although compliant), and α_4 not compliant (although safe).

As clear from the example, there might exist different correct allocations for a given set of resources and a given set of services. In the reminder of this section, we introduce the notion of cost of an allocation, to identify allocations that are correct and minimize the economic cost (*minimal* allocations).

7.2 Minimal allocation

Multiple factors contribute to the cost of an allocation (e.g., storage, data transfer, data computation). Since our reference scenario is one where the data owner is looking for a cloud service with basic support of access functionality, we consider storage and data transfer costs only. Our model can however be easily extended to consider different/additional cost factors. Figure 11 illustrates the costs, expressed in USD¢/GB, for our running example.

Storage. Storage cost is computed by considering the size of the resources to be outsourced. As noted in Section 6, we assume an estimation of the storage space requested for each (plaintext/encrypted) resource to be known by the owner. The storage cost is therefore computed by multiplying the size of the resources (in GB) by the storage price *st_cost*(s) (in GB) that is applied by cloud service s. The storage cost of an allocation α is then computed as follows:

$$\sum_{r \in \mathcal{R}} size^{\alpha(r)[m]}(r) \cdot st\_cost(\alpha(r)[s]) \qquad (1)$$

where $size^{\alpha(r)[m]}(r)$ denotes the size of resource r in the form m (plaintext/encrypted) in which r is allocated by α, and *st_cost*$(\alpha(r)[s])$ denotes the storage cost of the service s to which r is allocated by α. For instance, the storage cost of α_1 in Figure 8 is computed as:

$$\begin{aligned}&size^{\circ}(\texttt{admin})\cdot st\_cost(s_1)+\\&+size^{\bullet}(\texttt{agreements})\cdot st\_cost(s_3)+\\&+size^{\circ}(\texttt{suppliers})\cdot st\_cost(s_3)+\\&+size^{\circ}(\texttt{projects})\cdot st\_cost(s_1)+\\&+size^{\bullet}(\texttt{archive})\cdot st\_cost(s_3) = \text{USD } 7610¢.\end{aligned}$$

Data transfer. Data transfer cost is composed of in-bound and out-bound costs. In-bound traffic is usually free of charge while out-bound traffic has a cost that is typically proportional to the amount of data transferred, that is, to the size of accessed resources. The data transfer cost of an allocation is then obtained by multiplying the amount of out-bound traffic (in GB) from a cloud service (given by resources' size), by the corresponding data transfer cost *tr_cost*(s), weighted by the frequency with which resources are accessed (indeed, if a resource is never accessed, the cost of its transfer should not impact the total cost of an allocation). The data transfer cost of an allocation α is then computed as follows:

$$\sum_{r \in \mathcal{R}} size^{\alpha(r)[m]}(r) \cdot tr\_cost(\alpha(r)[s]) \cdot f(r) \qquad (2)$$

where $f(r)$ is the expected frequency of accesses to resource r. For instance, the data transfer cost of α_1 in Figure 8 is computed as:

$$\begin{aligned}&size^{\circ}(\texttt{admin})\cdot tr\_cost(s_1)\cdot f(\texttt{admin})+\\&+size^{\bullet}(\texttt{agreements})\cdot tr\_cost(s_3)\cdot f(\texttt{agreements})+\\&+size^{\circ}(\texttt{suppliers})\cdot tr\_cost(s_3)\cdot f(\texttt{suppliers})+\\&+size^{\circ}(\texttt{projects})\cdot tr\_cost(s_1)\cdot f(\texttt{projects})+\\&+size^{\bullet}(\texttt{archive})\cdot tr\_cost(s_3)\cdot f(\texttt{archive}) = \text{USD } 766.75¢.\end{aligned}$$

The cost of a correct allocation α, denoted *cost*(α), is obtained summing its storage and data transfer costs (Equations 1 and 2). As an example, consider the size and access frequencies of the resources in Figure 10 and the storage and transfer costs in Figure 11. The total cost of allocation α_1 in Figure 8 is *cost*$(\alpha_1) = 7610$ (storage cost) $+\ 766.75$ (data transfer cost) = USD 8376.75¢.

Formally, the problem of computing a correct allocation (Definition 7.2) with minimum economic cost is defined as follows.

Problem 7.1 (Minimal allocation). Let $\mathcal{R}$ be a set of resources, $\mathcal{S}$ be a set of cloud services, and $\langle\langle\lambda^{\circ}, \lambda^{\bullet}\rangle, \langle \mathsf{CoL}, \mathsf{Sep}, \mathsf{Usage}\rangle\rangle$ be an allocation policy. Determine a correct (Definition 7.2) allocation function α such that $\nexists\alpha' \neq \alpha$ that is correct and *cost*(α')<*cost*(α).

Allocation α_1 in Figure 8, with a cost of USD 8376.75¢, is an example of a minimal allocation for the resources in Figure 5 over the services in Figure 1. Allocation α_2, although correct, is not minimal since *cost*(α_2)=8520.50 >*cost*(α_1)=8376.75. Note that, while the minimum cost is unique, several minimal allocations (with the same minimum cost) may exist. All of them are equivalent with respect to requirements satisfaction and economic cost, and are a solution to the problem.

8 Computing a minimal allocation

To compute a minimal allocation, we translate Problem 7.1 into a binary programming problem that can be formulated as follows: given a *set of variables* that can take values in

	p_{ij}													e_{ij}
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7
admin	1													
agreements										1				
suppliers			1											
projects	1													
archive										1				

(a) Allocation α_1

	p_{ij}													e_{ij}
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7
admin								1						
agreements										1				
suppliers			1											
projects	1													
archive										1				

(b) Allocation α_2

Figure 12. Values assigned to p_{ij} and e_{ij} modeling allocations α_1 (a) and α_2 (b) in Figure 8

	$\hat{p}_{ij}$													$\hat{e}_{ij}$
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7
admin	1	1						1	1					
agreements	1	1	1				1	1	1	1	1		1	1
suppliers	1	1	1											
projects	1													
archive									1	1	1		1	1

Figure 13. Values assigned to $\hat{p}_{ij}$ and $\hat{e}_{ij}$ modeling safe allocations

{0,1}, a *set of constraints* over them, and an *objective function*, find an assignment of values to variables that satisfies all the constraints and that minimizes the value of the objective function. We now describe how the translation works by showing the set of variables, the constraints, and the objective function of the corresponding binary programming problem.

The allocation function $\alpha : \mathcal{R} \rightarrow \mathcal{S} \times \{\circ, \bullet\}$ can be interpreted as a function that assigns 0 or 1 to a set of variables. These variables model the allocation of each resource $r \in \mathcal{R}$ to a cloud service $s \in \mathcal{S}$, and the choice between plaintext and encrypted format of r. For each resource r_i and each cloud service s_j, we define two binary variables p_{ij} and e_{ij} modeling the allocation of r_i at s_j in plaintext or encrypted form, respectively:

$$p_{ij} = \begin{cases} 1 & \text{if } r_i \text{ is allocated plaintext at } s_j \\ 0 & \text{otherwise} \end{cases}$$

$$e_{ij} = \begin{cases} 1 & \text{if } r_i \text{ is allocated encrypted at } s_j \\ 0 & \text{otherwise} \end{cases}$$

A solution to the binary programming problem is an assignment of values to variables p_{ij} and e_{ij} that satisfies the constraints illustrated in the following, to guarantee that the assignment represents a correct allocation. Figures 12(a) and 12(b) illustrate the assignment of values to variables p_{ij} (left-hand side of the figure) and e_{ij} (right-hand side of the figure, with gray background to recall our use of gray color to indicate allocations of encrypted resources) for allocations α_1 and α_2, respectively, in Figure 8 (for readability, we only reported value 1, empty cells have value 0).

Allocation. An allocation assigns each resource, in either plaintext or encrypted form, to exactly one cloud service (i.e., the number of services to which each resource is allocated is exactly one). This translates to the following constraint:

$$\sum_{j=1}^{|\mathcal{S}|} p_{ij} + e_{ij} = 1, \ \forall i = 1, \ldots, |\mathcal{R}|$$

Intuitively, for each resource r_i, the constraints sums p_{ij} and e_{ij} over all the services. If the sum is equal to 1, then there exists only one service s_j such that p_{ij} (or e_{ij}) is equal to 1, meaning that r_i is allocated to s_j in plaintext (or encrypted).

Safe allocation. The satisfaction of the protection requirements in an allocation policy implies that each resource r in $\mathcal{R}$ is allocated to a cloud service whose security class dominates the one specified for the resource, according to the allocation mode (plaintext or encrypted). We then represent the cloud services to which resources can be allocated without violating the protection requirements (i.e., such that the allocation would be safe) through two binary variables $\hat{p}_{ij}$ and $\hat{e}_{ij}$ for each resource r_i and each service s_j. These variables represents whether a safe allocation would allocate (value 1) or not (value 0) r_i to cloud service s_j in plaintext ($\hat{p}_{ij}$) or in encrypted form ($\hat{e}_{ij}$). Figure 13 represents, for each resource r_i (on the rows) and each service s_j (on the columns) of our running example, the combinations for which $\hat{p}_{ij}$ (left-hand side) and $\hat{e}_{ij}$ (right-hand side) assume value 1. As visible from the figure, $\hat{p}_{ij}$ and $\hat{e}_{ij}$ represent all the possible safe (plaintext and encrypted) allocations for our running example illustrated in Figure 7 ($\hat{p}_{ij}$=1 for $\circ$, $\hat{e}_{ij}$=1 for $\bullet$). We note that, since multiple safe allocations can exist for each resource, each row of the figure can have more than one cell with value 1. The allocation of resources to services must therefore be in accordance with the values of these variables, meaning that a resource r_i can be allocated (in plaintext or encrypted, respectively) to s_j only if $\hat{p}_{ij}$=1 or $\hat{e}_{ij}$=1, respectively. This requirement is translated as follows:

$$\sum_{j=1}^{|\mathcal{S}|} (p_{ij} \cdot \hat{p}_{ij}) + (e_{ij} \cdot \hat{e}_{ij}) = 1, \ \forall i = 1, \ldots, |\mathcal{R}|$$

The product $p_{ij} \cdot \hat{p}_{ij}$ ($e_{ij} \cdot \hat{e}_{ij}$, respectively) has value 1 only if both p_{ij} and $\hat{p}_{ij}$ (e_{ij} and $\hat{e}_{ij}$, respectively) are equal to 1, that is, only if r_i is allocated in plaintext (encrypted, respectively) to a legitimate cloud service s_j. If the sum of these products, j=1, …, $|\mathcal{S}|$, is equal to 1, then each resource is allocated to a service satisfying the protection requirements, meaning that the allocation is safe.

Compliant allocation. We now show how the global allocation requirements can be translated to a set of constraints that a solution of the binary programming problem must satisfy.

- *Co-location requirement*. A co-location requirement is satisfied when all the resources involved in the requirement are allocated to the same cloud service. This requirement can then be translated as follows:

$$\sum_{j=1}^{|\mathcal{S}|} \left(\prod_{r_i \in loc} (p_{ij} + e_{ij}) \right) = 1, \forall loc \in \mathsf{CoL}$$

Given a resource r_i and a service s_j, $p_{ij} + e_{ij}$ is equal to 1 only if r_i is allocated, plaintext or encrypted, to

			$\hat{p}_{ij}$													$\hat{e}_{ij}$			
	f	$size^{\circ}$	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_1	s_2	s_3	s_4	s_5	s_6	s_7	$size^{\bullet}$	f	
admin	0.15	25	1	1						1	1						27.5	0.15	admin
agreements	0.15	15	1	1	1				1	1	1	1	1		1	1	16.5	0.15	agreements
suppliers	0.15	20	1	1	1												30	0.15	suppliers
projects	0.50	10	1														11	0.50	projects
archive	0.05	100									1	1	1		1	1	110	0.05	archive
		st_cost	50	50	40	50	10	80	50	50	50	40	50	10	80	50	*st_cost*		
		tr_cost	50	40	30	40	10	100	40	50	40	30	40	10	100	40	*tr_cost*		

Figure 14. Input of the binary programming problem

s_j. Hence, the internal product is equal to 1 for the cloud service that stores all the resources in the co-location requirement loc. If the sum over all services for these products is equal to 1, then the requirement is satisfied by (exactly) one service.

- *Separation requirement*. A separation requirement is satisfied when the involved resources are not all allocated to the same service in plaintext. In other words, there must exist at least one resource in the constraint encrypted or allocated to a different service. This requirement can be translated as follows:

$$\sum_{j=1}^{|\mathcal{S}|}\left(\prod_{r_i \in sep} p_{ij}\right) = 0, \forall sep \in \mathsf{Sep}$$

The internal product is equal to 1 when all resources involved in a separation requirement are managed by the same cloud service s_j and they are all in plaintext, meaning that the separation requirement is violated. Otherwise, if at least a resource is missing or is encrypted, the internal product is equal to 0. Hence, if the sum over all services for these products is equal to 0, no service violates the constraint.

- *Service usage*. A service usage requirement is satisfied when: *1)* the number of cloud services involved in the allocation is less than *max_services*; and *2)* the total size of the resources allocated to a cloud service is at least *min_storage*. Since the number of resources allocated to a cloud service s_j can be computed as $\sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij})$, the first condition can be translated as follows:

$$\sum_{j=1}^{|\mathcal{S}|}\left\lceil \frac{\sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij})}{1+\sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij})}\right\rceil \leq max\_services$$

Intuitively, to identify the number of services to which at least a resource has been allocated (either in plaintext or encrypted), the above formula divides the number of resources allocated to a service s_j by itself. Note that we add 1 to the denominator (and then round up the fraction) to ensure that the fraction is always computable, also for cloud services that do not manage any resource (i.e., for which the sum of $p_{ij}+e_{ij}$ over all the resources is equal to 0).
The second condition is instead translated as follows:

$$\sum_{i=1}^{|\mathcal{R}|}(size^{\circ}(r_i)\cdot p_{ij}) + (size^{\bullet}(r_i)\cdot e_{ij}) \geq min\_storage,$$

$$\forall j = 1, \ldots, |\mathcal{S}| : \sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij}) \geq 1$$

The product $size^{\circ}(r_i)\cdot p_{ij}$ ($size^{\bullet}(r_i)\cdot e_{ij}$, respectively) is equal to $size^{\circ}(r_i)$ ($size^{\bullet}(r_i)$, respectively) only if r_i is allocated to s_j. Hence, the sum computes the overall storage at each service. Clearly, the constraint should be satisfied only by services storing at least a resource (i.e., for which the sum of $p_{ij}+e_{ij}$ over all the resources is at least 1).

Objective function. The objective function of our binary programming problem, which needs to be minimized, models the cost of the allocation represented by variables p_{ij} and e_{ij}. The storage and transfer costs illustrated in Section 7.2 are therefore computed considering the values of variables p_{ij} and e_{ij}, to guarantee that the binary programming problem is equivalent to Problem 7.1.

- *Storage cost*. The storage cost is translated as follows:

$$\mathit{ST\_cost} = \sum_{j=1}^{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{R}|}((size^{\circ}(r_i)\cdot p_{ij}) + (size^{\bullet}(r_i)\cdot e_{ij}))\cdot$$
$$\cdot \mathit{st\_cost}(s_j)$$

where the storage cost *st_cost*(s_j) of cloud service s_j is multiplied by the size of the (plaintext or encrypted) resources actually allocated to s_j (i.e., $size^{\circ}(r_i)\cdot p_{ij} + size^{\bullet}(r_i)\cdot e_{ij}$).

- *Data transfer cost*. The data transfer cost is translated as follows:

$$\mathit{TR\_cost} = \sum_{j=1}^{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{R}|}((size^{\circ}(r_i)\cdot p_{ij}) + (size^{\bullet}(r_i)\cdot e_{ij}))\cdot$$
$$\cdot f(r_i)\cdot \mathit{tr\_cost}(s_j)$$

where the data transfer cost of cloud service s_j is multiplied by the global amount of out-bound traffic from s_j, which is the product of the size of the resources allocated to s_j and the frequency of accesses.

Figure 14 graphically illustrates, in tabular form, the parameters for the computation of a minimal allocation for our running example. Vectors *f*, $size^{\circ}$ and $size^{\bullet}$ report the frequencies of access and the size of (plaintext and encrypted) resources, which are reported on the rows of the figure. Vectors *st_cost* and *tr_cost* on the bottom of the figure represent the storage and transfer costs of the services (on the columns of the figure). Cells with 1 in the central matrices represent safe allocations for our running example (modeled by value 1 assigned to values $\hat{p}_{ij}$ and $\hat{e}_{ij}$ like in Figure 13, also reported here for readability). This graphical representation helps visualizing the process for computing the cost of an allocation, which graphically consists in sums

$$
\begin{array}{lll}
\mathbf{min} & \sum_{j=1}^{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{R}|}((size^{\circ}(r_i) \cdot p_{ij}) + (size^{\bullet}(r_i) \cdot e_{ij})) \cdot st\_cost(s_j) + & \texttt{// storage cost} \\
 & \sum_{j=1}^{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{R}|}((size^{\circ}(r_i) \cdot p_{ij}) + (size^{\bullet}(r_i) \cdot e_{ij})) \cdot f(r_i) \cdot tr\_cost(s_j) & \texttt{// transfer cost} \\
\mathbf{s.t.} & \sum_{j=1}^{|\mathcal{S}|} p_{ij} + e_{ij} = 1,\ \forall i = 1, \ldots, |\mathcal{R}| & \texttt{// allocation function} \\
 & \sum_{j=1}^{|\mathcal{S}|}(p_{ij} \cdot \hat{p}_{ij}) + (e_{ij} \cdot \hat{e}_{ij}) = 1,\ \forall i = 1, \ldots, |\mathcal{R}| & \texttt{// protection requirements} \\
 & \sum_{j=1}^{|\mathcal{S}|} \left(\prod_{r_i \in loc}(p_{ij} + e_{ij})\right) = 1, \forall loc \in \mathsf{CoL} & \texttt{// co-location requirements} \\
 & \sum_{j=1}^{|\mathcal{S}|} \left(\prod_{r_i \in sep} p_{ij}\right) = 0, \forall sep \in \mathsf{Sep} & \texttt{// separation requirements} \\
 & \sum_{j=1}^{|\mathcal{S}|} \left\lceil \frac{\sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij})}{1+\sum_{i=1}^{|\mathcal{R}|}(p_{ij}+e_{ij})} \right\rceil \leq max\_services & \texttt{// service usage requirement} \\
 & \sum_{i=1}^{|\mathcal{R}|}(size^{\circ}(r_i) \cdot p_{ij}) + (size^{\bullet}(r_i) \cdot e_{ij}) \geq min\_storage, \forall j = 1, \ldots, |\mathcal{S}| : \sum_{i=1}^{|\mathcal{R}|}(p_{ij} + e_{ij}) \geq 1 & \texttt{// service usage requirement}
\end{array}
$$

Figure 15. Binary programming formulation of Problem 7.1

and multiplications among vectors (representing resource sizes, access frequencies, and service costs) and matrices (representing safe allocations). For instance, the transfer cost of allocating `admin` to s_1 in plaintext requires to multiply the frequency of `admin` (0.15) by its size (25GB) by the transfer cost of s_1 (USD 50¢/GB), as reported in the figure. Note that the cost of unsafe allocations (i.e., those for which the cells of the central matrices do not contain value 1) is automatically discarded by our approach, thanks to the constraint imposed on the satisfaction of the protection requirements that considers only those allocation for which $\hat{p}_{ij}$ or $\hat{e}_{ij}$ are equal to 1.

Having defined how the costs of an allocation can be computed, the objective function of our binary programming problem is formulated as follows:

$$\mathbf{min}(\mathit{ST\_cost} + \mathit{TR\_cost})$$

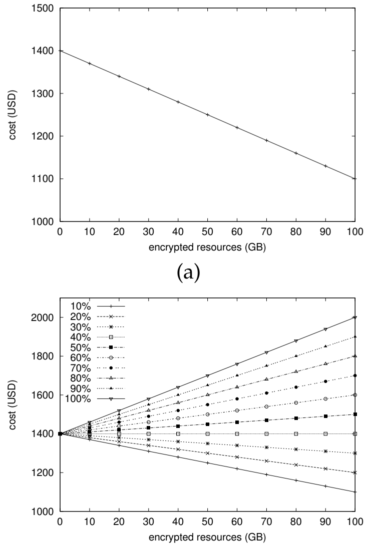
Figure 15 summarizes the formulation of a binary programming problem equivalent to Problem 7.1. The allocation in Figure 12(a) illustrates the output (in terms of value assignment to variables p_{ij} and e_{ij}) of the problem in Figure 15 for the running example, corresponding to the minimal allocation α_1 in Figure 8.

9 Discussion

We conclude the presentation of our solution with some observations related to the use of encryption at the data owner side. In principle, the encryption of resources by the data owner can have two main advantages: *i)* flexibility in the allocation, especially with respect to the enforcement of the separation requirements, and *ii)* savings in the economic cost of the allocation. The first advantage arises from the fact that the satisfaction of a separation requirement can be guaranteed by encrypting at least one of the resources involved. The encryption may then allow the management of all resources in a separation requirement by (possibly) the most convenient cloud service. For instance, consider the separation requirement $\{$`agreements`,`suppliers`$\}$ of our running example. To correctly enforce it, the data owner can encrypt resource `agreements` and allocate both `suppliers` and (encrypted) `agreements` to s_3, with a cost of USD 8376.75¢ (i.e., as in the minimal allocation for our running example). Alternatively, if we suppose that the resources cannot be encrypted, the only way for correctly enforcing the separation requirement consists in allocating the resources to two different cloud services. In this case, a minimal allocation would allocate `suppliers` to s_2 and `agreements` to s_3, with a (higher) cost of USD 8540¢.

The second advantage (economical savings) arises from the fact that in general, provided an expected correlation between the cost of a service and the guarantees it offers (meaning that, as often happens in real life scenarios, more expensive services provide higher security guarantees), owner-side encryption can enable the involvement of less trusted and more affordable services for storing a portion of the resources. To validate this observation, we have considered a case study with 10 resources of 10GB each with equal access frequency, and two cloud services s_a and s_b with cost USD 10¢/GB and USD 14¢/GB respectively (where the cost sums the storage and transfer since accesses to all resources have the same frequency). For the case study, we assume that s_a can store only encrypted resources, while s_b is trusted to store plaintext resources. Figure 16(a) illustrates the cost of an allocation varying the amount of resources that the data owner is willing to encrypt, from 0GB (i.e., no encryption) to 100GB (i.e., all resources can be encrypted), assuming a 10% increase in the size of resources caused by encryption. The figure clearly shows that, increasing the amount of encrypted resources, the cost of the allocation decreases. This is due to the possibility of storing more resources to the more affordable s_a.

However, different encryption schemes might impact the cost of an allocation in different ways. In extreme cases, it might happen that the use of a cheaper cloud service for storing encrypted resources results in non-minimal allocations, where the increase in the cost is due to a substantial increase in the size of the encrypted resources. We have evaluated the impact of the size increase caused by encryption on the case study above, evaluating the cost of different allocations storing encrypted resources on s_a (less trusted and less expensive) and plaintext resources on s_b (more trusted and more expensive), varying the amount of resources that can be encrypted (from 0GB to 100GB) and the size increase (from 10% to 100%). The results are reported in Figure 16(b). In this case study, encryption can help reduce the cost of allocations when encryption causes an increase in the size of resources up to 30%, meaning that it is actually more convenient to store as many encrypted resources as possible to s_a. For an increase of 40%, the cost remains stable regardless of the amount of encrypted resources, while for larger increases storing encrypted resources at s_a would be more costly than keeping the resources at s_b. Note that our approach automatically takes into account all



(a)

(b)

Figure 16. Effects of encryption on allocation cost varying the amount of encrypted resources (a), and varying the amount of encrypted resources and their size increase due to encryption (b)

these considerations, encrypting resources only if necessary to satisfy requirements and if the resulting allocation has minimum cost.

10 Related work

Substantial effort has been recently devoted to the definition of multi-cloud storage systems, possibly taking security and cost into account. The majority of the proposals however aim at leveraging the availability of multiple providers/services to store fragments or replicas of a data collection, while we aim at supporting (fine-grained) protection and global requirements to find an optimal allocation of resources to services, without fragmenting or replicating data. For example, the proposal in [5] splits data among cloud providers based on a pre-defined budget with the goal of involving no less than a certain number of providers for the successful retrieval of a given data item. The storage system in [6] considers data confidentiality, integrity, and availability, which are also modeled by our security properties, but aims at ensuring them replicating data on different providers, and does not account for different requirements over different data items (while we support fine-grained protection requirements at the level of single resource or classes thereof). The proposal in [7] splits data among providers through erasure codes, and (re-)allocates data chunks to providers depending on how the chunks are accessed over the time and on the physical location and economic cost of the allocation itself. Similarly, the works in [4], [8], [9], [10] propose multi-cloud storage systems where data are replicated/fragmented across providers (differently from our approach that neither replicates nor fragments data). In [11], the authors propose a framework for monitoring multi-cloud storage platforms and adapting their infrastructure to environmental changes (e.g., failures), while we specifically aim at supporting multi-cloud data allocation according to users' requirements.

Another related line of work addresses the problem of selecting an optimal (set of) provider(s) according to a given policy. In [12], the authors propose an automated approach for selecting storage providers based on user requirements, focusing on cost and performances factors. Our work, while related, provides a security model allowing a data owner to formulate requirements guiding the allocation process, and defines an approach for computing an optimal allocation satisfying all the identified constraints. The works in [13], [14], [15] adopt fuzzy logic and reasoning for cloud providers selection. In particular, fuzzy logic is adopted in [15] for supporting users in specifying high-level requirements with natural language. The approach in [1] proposes a language for expressing user requirements and preferences, a formal model for reasoning on them, and different strategies for ranking acceptable services. While sharing with our problem the idea of a user-centered selection approach, our work pursues a different goal related to finding an optimal allocation of resources to providers by considering both protection and global requirements. In this regard, our approach can build on the proposals in [1] and [15] for requirements specification. The proposal in [16], while sharing with ours the goal of supporting the selection of multiple providers/services, assumes requirements as importance levels given to pre-defined Service Level Objectives (SLOs). Other related proposals address the problem of determining an optimal resource allocation considering providers load [17], fault tolerance aspects [18], providers' restrictions and users' placement constraints [19], the adoption of multi-cloud solutions for developing and managing applications [20], and the integration of multi-cloud storage with existing NAS-based programs [21].

Our support for user requirements show similarities with the approach in [22], proposing a CSP-based automated framework for composing SLAs based on application requirements and dependencies among the characteristics of a provider. The approaches in [2], [23] propose a consensus-based approach for accommodating contrasting requirements from multiple applications. While related, these proposals address a problem differing from ours.

11 Conclusions

We proposed an approach for allocating resources to cloud services. Our solution relies on the definition of a security model based on abstract security properties, over which cloud services can be classified and resource protection requirements can be specified. The allocation of resources to services, which even accounts for owner-side encryption, guarantees the satisfaction of such protection requirements, as well as of global allocation requirements that the data owner can specify to limit the overhead at her side and an excessive fragmentation of resources across multiple services. We propose a formulation of the problem in terms of a binary programming problem, to compute a correct allocation that minimizes economic costs.

Acknowledgments

This work was supported in part by: the EC within the H2020 Program under grant agreement 825333 (MOSAICrOWN), and the Italian MIUR PRIN project "HOPE".

References

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

[2] A. Arman, S. Foresti, G. Livraga, and P. Samarati, "Cloud plan selection under requirements of multiple applications," *Security and Privacy*, vol. 1, no. 4, July/August 2018.

[3] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Fragments and loose associations: Respecting privacy in data publishing," *PVLDB*, vol. 3, no. 1, pp. 1370–1381, September 2010.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Fragmentation in presence of data dependencies," *IEEE TDSC*, vol. 11, no. 6, pp. 510–523, November/December 2014.

[5] Y. Singh, F. Kandah, and W. Zhang, "A secured cost-effective multi-cloud storage in cloud computing," in *Proc. of IEEE INFOCOM WKSHPS 2011*, Shanghai, China, April 2011.

[6] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DEPSKY: Dependable and secure storage in a cloud-of-clouds," *ACM TOS*, vol. 9, no. 4, pp. 12:1–12:33, November 2013.

[7] T. G. Papaioannou, N. Bonvin, and K. Aberer, "Scalia: An adaptive scheme for efficient multi-cloud storage," in *Proc. of IEEE SC 2012*, Salt Lake City, UT, USA, November 2012.

[8] D. Bermbach, M. Klems, S. Tai, and M. Menzel, "MetaStorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proc. of IEEE CLOUD 2011*, Washington DC, USA, July 2011.

[9] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A case for cloud storage diversity," in *Proc. of ACM SoCC 2010*, Indianapolis, IN, USA, June 2010.

[10] R. M. d. O. Libardi, S. Reiff-Marganiec, L. H. Nunes, L. J. Adami, C. H. Ferreira, and J. C. Estrella, "MSSF: User-friendly multi-cloud data dispersal," in *Proc. of IEEE CLOUD 2015*, New York, NY, USA, June/July 2015.

[11] A. Rafique, D. Van Landuyt, V. Reniers, and W. Joosen, "Towards an adaptive middleware for efficient multi-cloud data storage," in *Proc. of CrossCloud 2017*, Belgrade, Serbia, April 2017.

[12] A. Ruiz-Alvarez and M. Humphrey, "A model and decision procedure for data storage in cloud computing," in *Proc. of CCGrid 2012*, Ottawa, Canada, May 2012.

[13] A. V. Dastjerdi and R. Buyya, "Compatibility-aware cloud service composition under fuzzy preferences of users," *IEEE TCC*, vol. 2, no. 1, pp. 1–13, January 2014.

[14] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory," *IEEE TC*, vol. 65, no. 8, pp. 2348–2362, August 2016.

[15] 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*, 2019 (to appear).

[16] A. Taha, S. Manzoor, and N. Suri, "SLA-based Service Selection for Multi-Cloud Environments," in *Proc. of EDGE 2017*, Honolulu, HI, USA, June 2017.

[17] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized server selection for cloud services," in *Proc. of ACM SIGCOMM 2010*, New Delhi, India, August/September 2010.

[18] R. Jhawar, V. Piuri, and M. Santambrogio, "Fault tolerance management in cloud computing: a system-level perspective," *IEEE Systems Journal*, vol. 7, no. 2, pp. 288–297, June 2013.

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

[20] N. Ferry, F. Chauvel, H. Song, A. Rossini, M. Lushpenko, and A. Solberg, "CloudMF: Model-Driven management of multi-cloud applications," *ACM TOIT*, vol. 18, no. 2, pp. 16:1–16:24, January 2018.

[21] M. Chen and E. Zadok, "Kurma: Secure geo-distributed multi-cloud storage gateways," in *Proc. of ACM SYSTOR 2019*, Haifa, Israel, June 2019.

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

[23] A. Arman, S. Foresti, G. Livraga, and P. Samarati, "A consensus-based approach for selecting cloud plans," in *Proc. of IEEE RTSI 2016*, Bologna, Italy, September 2016.

Sabrina De Capitani di Vimercati is a professor at the Computer Science Department, Università degli Studi di Milano, Italy. Her research interests are in security, privacy, and data protection. She has published more than 210 papers in journals, conference proceedings, and books. She has been a visiting researcher at SRI International, CA (USA), and George Mason University, VA (USA).
http://www.di.unimi.it/decapita

Sara Foresti is a professor at the Computer Science Department, Università degli Studi di Milano, Italy. Her research interests are in the area of data security and privacy. She has published more than 100 papers in journals, conference proceedings, and books. She has been a visiting researcher at George Mason University, VA (USA). She chairs the IEEE CS Italy Chapter.
http://www.di.unimi.it/foresti

Giovanni Livraga is an assistant professor at the Computer Science Department, Università degli Studi di Milano, Italy. His research interests are in the area of data privacy and security in emerging scenarios. His PhD thesis received the ERCIM STM WG 2015 award. He has been a visiting researcher at SAP Labs, France and George Mason University, VA (USA).
http://www.di.unimi.it/livraga

Vincenzo Piuri is a professor at the Computer Science Department, Università degli Studi di Milano, Italy. His main research interests are signal and image processing, biometrics, intelligent systems, digital and signal processing architectures. He has published more than 350 papers in journals, conference proceedings, and books. He has been named IEEE Fellow (2001) and ACM Distinguished Scientist (2008).
http://www.di.unimi.it/piuri

Pierangela Samarati is a professor at the Computer Science Department, Università degli Studi di Milano, Italy. Her main research interests are in data protection, security, and privacy. She has participated in/coordinated several projects at national and international level. She has published more than 270 papers in journals, conference proceedings, and books. She has been named ACM Distinguished Scientist (2009) and IEEE Fellow (2012).
http://www.di.unimi.it/samarati