Project title: Enforceable Security in the Cloud to Uphold Data Ownership
Project acronym: ESCUDO-CLOUD
Funding scheme: H2020-ICT-2014
Topic: ICT-07-2014
Project duration: January 2015 – December 2017

D4.4
Report on Multi Cloud and Federated Cloud

Editors: Ahmed Taha (TUD)
Heng Zhang (TUD)
Neeraj Suri (TUD)
Reviewers: Björn Tackmann (IBM)
Daniel Bernau (SAP)

Abstract

This deliverable reports on the final version of the techniques designed and implemented to cover multi Cloud and federated Cloud across tasks T4.1, T4.2 and T4.3. It details the activities conducted on the definition and selection of trust metrics in T4.1, the development of the techniques for leveraging multiple providers for security and efficiency in T4.2, and defines the federated secure Cloud storage service supporting proper data protection in T4.3.

Type	Identifier	Dissemination	Date
Deliverable	D4.4	Public	2017.10.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644579. This work was supported in part by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract No 150087. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or the Swiss Government.

ESCUDO-CLOUD Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	British Telecom	BT	United Kingdom
3.	EMC Corporation	EMC	Ireland
4.	IBM Research GmbH	IBM	Switzerland
5.	SAP SE	SAP	Germany
6.	Technische Universität Darmstadt	TUD	Germany
7.	Università degli Studi di Bergamo	UNIBG	Italy
8.	Wellness Telecom	WT	Spain

Versions

Version	Date	Description
0.1	2017.10.05	Initial Release
0.2	2017.10.26	Second Release
1.0	2017.10.31	Final Release

List of Contributors

This document contains contributions from different ESCUDO-CLOUD partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	All Partners
Chapter 1: Security metrics and assessment	Ahmed Taha (TUD), Heng Zhang (TUD), Neeraj Suri (TUD)
Chapter 2: Multiple providers for security	Sara Foresti (UNIMI), Giovanni Livraga (UNIMI), Andrew Byrne (EMC), Alan Barnett (EMC), Enrico Bacis (UNIBG), Marco Rosa (UNIBG)
Chapter 3: Data Protection as a Service for Federated Cloud Storage	Ali Sajjad (BT)
Chapter 4: Conclusion	All Partners

Contents

List of Figures

Executive Summary

This deliverable provides an integrated view of the contributions and findings that were produced by Tasks 1 to 3 under Work Package 4 (WP4). The aim of WP4 within ESCUDO-CLOUD is to develop solutions for allowing users to leverage the availability of multiple providers, to enhance security and reduce costs. The presence of multiple providers is utilized for developing a secure Cloud storage service. As mentioned in the DoA, the scope of WP4 entailed developing techniques to conduct trust quantification and assessment, developing techniques for leveraging multiple providers for enhanced security and efficiency, and finally to develop a federated secure Cloud service supporting the desired levels of data protection. For achieving these objectives, the work was conducted over three tasks:

T4.1 Security metrics and assessment (M1-M24). The task developed techniques to conduct security assessment based on the quantitative and qualitative analysis of security information. It proposes techniques to conduct quantification of information, techniques for information composition and for security comparisons across Cloud providers. The task, covering single, multi and federated Clouds, allows users to reason about security risks.

T4.2 Multiple providers for security (M7-M34). The task developed techniques that leverage the presence of multiple providers to enhance security, while also maximizing efficiency of accesses. It investigates the implementation of the distributed shuffle index (originally developed in WP2) with different Cloud providers and its integration with real industrial scenarios. The task analyzes distributed Cloud storage and All-or-Nothing-Transform encryption mode for providing security. In addition, the task proposes a model for expressing user requirements and preferences to be enforced in the selection of Cloud services in multi-provider scenarios.

T4.3 Data Protection as a Service for Federated Cloud Storage (M7-M34). The task developed and integrated techniques and components for protecting data-at-rest using the federated Data Protection as a Service solution for different Cloud storage services offered on multiple Cloud platforms. In addition to that, the task highlighted the salient functions and features of the components of the solution, their purpose and motivation in the architecture, and their role within the scope of Use Case 3.

These solutions will be summarized within this deliverable by their corresponding tasks with varying levels of coverage. Where the content has already been presented in prior deliverables, a summary of the highlights is presented. The new content will be featured in higher detail. The content of WP4 is directly built upon the ESCUDO-CLOUD UCs from WP1. Also, several technology transfers of the results developed in WP4 have already been realized by the ESCUDO-CLOUD partners.

The remainder of this document is organized as follows. Chapter 1 summarizes the work from task T4.1 that ended at M24. Chapter 2 presents the work conducted in T4.2. Chapter 3

summarizes the activities conducted in T4.3. Lastly, Chapter 4 presents the overall conclusions on the findings of WP4 along with directions of future work.

1. Security metrics and assessment

Task 4.1 finished at M24 and this section reports a summary of the activities conducted therein over D4.1 and D4.2. The work under Task 4.1 focused on outlining the security metrics applicable to assessing the security services offered by different Cloud Service Providers (CSPs), as per their Service Level Agreements (SLAs). The goal is to allow Cloud users to quantitatively evaluate CSP SLAs to attain security assurance. D4.1 developed two evaluation techniques for assessing the security levels provided by different CSPs as per their SLAs. Subsequently D4.2 extended the techniques to the Multi-Cloud environment. In the following, Section 1.1 presents an overview of the innovations within Task 4.1. Section 1.2 summarizes the work reported in D4.1 and D4.2.

1.1 ESCUDO-CLOUD Innovation

This task produced several advancements over the state of the art.

- *Extended QPT and QHP.* The elicitation of security requirements needed to quantify and aggregate the security levels provided by different CSPs was conducted as applicable to ESCUDO-CLOUD Use Cases. Furthermore, two evaluation techniques, namely Quantitative Policy Trees (QPT) and Quantitative Hierarchical Process (QHP), were extended for conducting the quantitative assessment and analysis of the SLA based security levels provided by CSPs with respect to a set of user security requirements.

- *CSP Parameterization.* A sensitivity analysis based on Cloud SLAs was conducted for helping CSPs to (a) determine which parameter(s) most affects the overall security level (as per the Cloud user requirements), and (b) provide guidance on the security improvements that should be performed by the CSP in order to achieve the requested security level.

- *Dependency Analysis.* The evaluation of the service levels provided in Multi-Cloud services was conducted using a novel dependency analysis process developed in ESCUDO-CLOUD. In a Multi-Cloud environment, different Cloud providers might implement different metrics, and this can potentially result in conflicts across their SLA specifications. Hence, Task 4.1 developed a process to ascertain the dependencies between the metrics. The dependency analysis formed the basis to develop techniques for the accurate trust assessment of Multi-Cloud services.

1.2 Security Metrics and Assessment

T4.1 finished at M24 and the results were reported in D4.1 & D4.2 whose summary is presented in the subsequent subsections. As a note, while T4.1 ended at M24, its activities have been sustained internally and have resulted in the publications [LTTS15, MTT+16, TMT+16, TMS+17b, TTLS17, TMS17a].

1.2.1 D4.1 Summary - SLA Based Metrics and Assessment Approaches

The quantitative security-level assessment of CSPs based on SLAs (for their match to the user requirements) is the primary objective of the techniques developed in D4.1, namely the Quantitative Policy Trees (QPT) and the Quantitative Hierarchy Process (QHP). Using this assessment, the CSPs are ranked (as per their SLAs) for the best match to the user requirements. QPT utilizes a logical aggregation of security quantifiers, while QHP is based on multi-variable optimization techniques considering the various elements of a SLA as the optimization criteria.

As an overview of the two techniques, the SLA assessment and the ranking of CSPs are performed in three progressive stages common to both the QPT and QHP techniques. In Stage A, using a common syntax, we express both the user requirements and the CSP committed service level objectives (SLOs)[1] using a standardized SLA template (e.g., based on ISO/IEC 19086 [Int16]). In Stage B, the user requirements and CSP SLAs are quantitatively evaluated. This quantitative data is then used in Stage C as input to a ranking algorithm, in order to provide the final assessment result. In the subsequent sections, we provide a brief overview of the QPT and QHP techniques.

Quantitative Policy Trees (QPT)

Luna et al. [LLS12] proposed the use of a tree-like data structure (i.e., the Quantitative Policy Tree) to model each CSP's security policy in order to numerically evaluate it with respect to a set of user requirements.

Stage A: Definition of Security Requirements. The QPT is an AND/OR tree where the Cloud user's requirements are also represented as a security SLA (called *user SLA*). The service controls are represented as intermediate nodes of the tree, while the security metrics associated with SLOs are represented as leaf nodes. Assigned weights are used to represent the relative importance of SLOs from the user's perspective

To complete the customization of a *user SLA*, the user can also select the appropriate AND/OR relationships. The overall QPT creation process is shown in Figure 1.1.

Stage B: Security Quantification. In order to evaluate the *Cloud user SLA* (termed as the user QPT) with respect to the offered *CSP SLA* (CSP QPT), the QPT utilizes the notion of local security levels (LSL) [CPRT05] and two basic assumptions: *(i)* all the *i-leaf nodes* on the QPT have been already associated with a $LSL_i > 0$ and, *(ii)* there exists a maximum value LSL_{max} that is the same for *all* the leaf nodes of the QPT (i.e., $0 < LSL_i \leq LSL_{max}$).

Once each leaf node in the QPT has been associated with the tuple $\{LSL_i, \omega_i\}$, it is possible to propagate these values to the rest of the tree using the aggregation rules shown in Table 1.1.

Table 1.1: Aggregation rules for a QPT with n-sibling nodes [LLS12]

Parameter	Aggregation rule with $i = 1 \ldots n$	
	AND node	OR node
$Agg_{ParentL1}$	$\sum_{i=1}^{n}(LSL_i \times \omega_i)$	$min(LSL_i \times \omega_i)$
$Agg_{ParentL2}$	$\sum_{i=1}^{n} Agg_{ParentL1,i}$	$min(Agg_{ParentL1,i})$

Stage C: Security Evaluation. Once the *Cloud user QPT* and the *CSP QPT* have been populated with the aggregated values (quantitatively computed from Table 1.1), it is possible to apply

[1] SLOs are the measurable elements of an SLA which consist of different metric values V_i. Each value implies a different Cloud service leve requested by the users, and required to be achieved by the CSP.

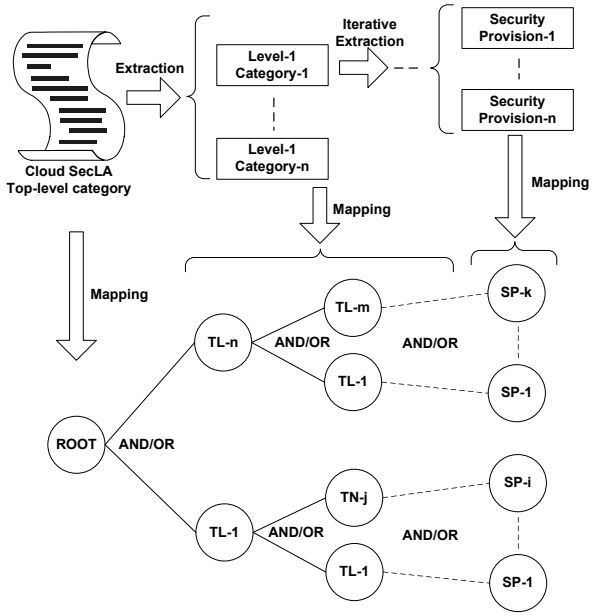

Figure 1.1: SLA-to-QPT: mapping a Cloud SLA into a QPT[LLS12].

a ranking process to determine how different CSPs under-/over-provision a user's requirement. Luna et al. [LLS12] proposed two different classes of benchmarks (quantitative and qualitative) based on the quantitative security values already aggregated in the QPT.

Quantitative Hierarchy Process (QHP)

The Quantitative Hierarchy Process (QHP) [TTLS14] allows users to (i) compare, benchmark and rank the aggregated security level provided by two or more CSPs according to the users' levels of security expertise, (ii) provide a composite quantitative and qualitative security assessment technique based on the well-known Analytic Hierarchy Process (AHP) [Saa90] (depending on the user defined security requirements and priorities), and (iii) automate the overall assessment process.

Similar to the QPT, the SLA assessment and ranking of CSPs is a ESCUDO-CLOUD developed extension of QHP [TTLS14], as developed in the progressive stages described below and depicted in Figure 1.2.

Stage A: Definition of Security Requirements. In this stage, the Cloud user creates his set of security requirements based on the same SLA template (structure) used by the CSPs to specify their security needs.

Stage B: Security Quantification. In order to evaluate the user requirements with respect to a CSP SLA, the measurement model for different security SLO metrics needs to be defined. The relationship across the CSPs with respect to each security SLO k is represented as a ratio:

$$CSP_{1,k}/CSP_{2,k} = \frac{V_{1,k}}{V_{2,k}}$$

where $CSP_{1,k}/CSP_{2,k}$ indicates the relative rank of CSP_1 over CSP_2 for SLO k. $V_{1,k}$ and $V_{2,k}$ specifies the service level offered by CSP_1 and CSP_2, for SLO k, respectively. The security SLO metrics under evaluation can be boolean (e.g., a yes/no representing the need of a security mechanism) or numerical (e.g., a cryptographic key length).

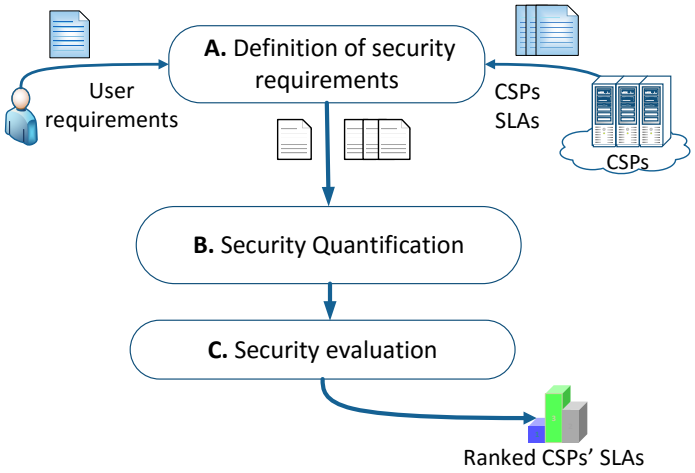

Figure 1.2: Stages comprising the quantitative SLA assessment.

Stage C: Security Evaluation. The challenge is not only how to quantify different metrics, but also how to aggregate them in a meaningful way. To solve these challenges, QHP's ranking mechanism was developed based on AHP [Saa90] for solving Multiple Criteria Decision Making (MCDM) [Zel82] problems. The AHP-based methodology for CSP rankings consists of four main steps: (1) hierarchy structure (2) weights assignment (3) pairwise comparison and (4) SLOs aggregation to give the overall rank calculation.

QPT and QHP Validation: Case Studies

D4.1 also conducted an empirical validation across QPT and QHP demonstrating the relative advantages and disadvantages of each approach as depicted in Table 1.2.

Table 1.2: QPT and QHP — comparison of main features

Stage	Feature	QPT	QHP
Security Requirements	SLA granularity for expressing requirements	Weights and values only at SLO level	Weights and values at all levels
	Supported SLO values	Quantitative and Qualitative	
	Template for user requirements	SLA hierarchy	
	Model relationships among SLA elements	AND/OR among SLOs	None
Security Quantification	Base technique for aggregation	Ad-hoc	Multi-criteria decision technique
	Used SLA abstraction	AND/OR Tree	Matrix
Security Evaluation	Output	Ranked List, Overall Security Level	
	Format of resulting security level	Quantitative/ Qualitative	Quantitative

The empirical validation is performed through two scenarios that use real world SLAs struc-

tured in compliance with the current draft version of the ISO/IEC 19086 standard [Int16] and with data derived from the Cloud Security Alliance's public STAR repository [STA11].

The validation scenarios were designed taking into account real concerns from Cloud users (i.e., procuring Cloud services based on security requirements) and CSPs (i.e., maximum offered security levels). The data set consisted of different combinations of requirements and real SLA representing three different users, and three different CSPs respectively.

Summary

The work of D4.1 was reported at M12 and resulted in the publications [LTTS15, TMT+16, MTT+16, TTLS17, TMS+17b]. Overall we were able to achieve the following:

- Address trust metrics and the SLA-based solutions for enabling users to express and reason about the trust associated with different providers for matching their requirements.
- Develop two evaluation techniques for conducting the quantitative assessment and analysis of the SLA based security level provided by the CSPs.
- Validate the two evaluation techniques using two use case scenarios and a prototype, leveraging actual real-world CSP SLA data derived from the publicly available CSA STAR.

1.2.2 D4.2 Summary - SLA Based Metrics and Assessment in the Multi-Cloud

Despite the frequently advocated economic and performance-related advantages of utilizing multiple Clouds, two basic concerns are not yet fully addressed in the community. First, with the growth of public Cloud services, multiple CSPs offer "similar" services at different prices and capabilities. Second, most of these services are typically bundled together with both explicit and implicit dependency relations[2] across them. For example, Dropbox (as an SaaS provider) depends on (IaaS providers) Amazon's S3 service for storage and on EC2 for computation [DMM+12] to support its SaaS software services.

Another example is the "encryption keys management" service which depends on multiple factors. Firstly, on the specific techniques used to store the encryption keys. Secondly, on the processes specifying how keys are accessed and the possibility of the key recovery. Finally, it depends on the control and management of each key. Each of these factors contains different levels of services (e.g., different techniques to store and distribute the keys) which the user can require and the CSP agrees to fulfill. Most of these factors are dependent on each other [TMT+16]. These dependency relations can easily introduce conflicts. For instance, a user may require an unachievable level of a dependent service which can result in the user requirements to be impossible to satisfy. The questions remain as to how users can (a) score and then select services for each single requirement, and (b) how to optimize this allocation to satisfy the requirements of the composite service, taking into consideration that these concerned services can be conflicting or have different degree of importance for the user.

D4.2 presented a quantitative service-level assessment of CSPs, in a Multi-Cloud environment, to find the service composition that collectively satisfies all the user requirements. The selection of composite services and the dependency management for composing the services is performed in five main stages. After the CSPs submit their SLAs and the users specify their requirements in

[2]Dependency relations between services or simply service dependencies are the direct relations between one or more services, where a service can depend on data or resources provided by another service.

Stage A, the services assessment and selection algorithms are used to score services in Stage B according to the user requirements. Based on this assessment, an optimum combination of services from multiple CSPs is chosen. This combination is used to create the MCSLA (Multi-Cloud SLA) in Stage C. Based on the MCSLA services allocation, a dependency model is created in Stage D to capture information about the composite services and the dependencies that occur between them. This model is specified using a machine readable format to allow automated validation for checking service conflicts and SLO compatibility issues in Stage E.

Stage A: User Requirements Definition. During this stage, the users create their set of requirements and specify their preferences based on the same SLA template used by the CSPs to specify their offered services.

Stage B: Services Evaluation. The quantitative service level assessment of CSPs (for their match to the user requirements) is developed in this stage. Using this assessment, the services are scored based on the user requirement level for each service. The SLA services assessment is performed in five progressive phases. In *Phase 1* the SLAs are modeled as a hierarchical structure. In order to compare two CSP's SLOs, the relative importance levels of the user requirements for each SLO are assigned as weights in *Phase 2*. In *Phase 3* a measurement model for different SLOs is defined. Finding the best combination of SLOs that can collectively satisfy the user needs is performed in *Phase 4*. Furthermore, an overall assessment of the service levels using a bottom-up aggregation method is presented in *Phase 5*.

Stage C: Multi-Cloud Service Allocation. After finding the best combination of services that collectively satisfy all the user needs in Stage B, mapping this combination to the Multi-Cloud SLA (named MCSLA) is achieved in this stage. This is achieved by constructing the MCSLA template and then using the service scores provided in the previous stage to build the composition of the feasible services. The main purpose of constructing such an MCSLA is to address the SLA interoperability issue in a Multi-Cloud setting.

In order to automatically and quantitatively evaluate the MCSLA, the classical "AND-OR" trees is used to identify the set of services that satisfy the user requirements. This is done by forming compositions of services represented by each "AND" or "OR". As inferred from their name, "AND" relationships represent the "necessary" user requirements and "OR" relationships are more adequate to model "optional" requirements.

After mapping the selected services to the MCSLA, a dependency model is created in the next stage to capture the information about MCSLA services and also the dependencies that occur between them.

Stage D: Dependency Model Creation. The approach for managing service dependencies builds on a dependency model, which is used to capture information about various services (each composed of a set of SLOs) and the dependencies that occur between them. A meta-model is then developed based on the defined dependency model. The meta-model is specified using a machine readable format (allowing fully automatic validation) such as an XML data structure using an XML Schema.

Stage E: MCSLA Validation. Following the MCSLA and dependency model construction, the MCSLA is validated to check service conflicts and different SLOs compatibility issues. At the end of this stage a list of all conflicts found in the validated MCSLA with the conflict's explanation is created which allows these conflicts to be understood and resolved.

This developed 5-stage approach constituted the dependency analysis process to conduct accurate trust assessments in Multi-Cloud environments.

Summary

The work of D4.2 was reported at M24 and resulted in the publications [MTT+16, TMT+16, TMS+17b, TMS17a]. Overall we were able to achieve the following:

- Develop a novel SLA-based service selection methodology for the Multi-Cloud environment that (a) provides a MCSLA construction, and (b) evaluates the service levels provided in Multi-Cloud services, considering the dependencies among the services provided by its participants.
- Develop a novel SLO/SLA-level dependency analysis technique applicable to Multi-Cloud environments.
- Develop a comprehensive SLA-based quantitative trust assessment schema applicable to the Cloud and Multi-Clouds.

2. Multiple providers for security

The work under Task 4.2 has been devoted to the use of multiple providers for security. A first line of work has addressed the use of the shuffle index approach, originally developed in Work Package 2. In D4.2 and D4.3 we have focused on distributing the shuffle index among multiple providers to enhance its security guarantees and on implementing it through the integration with different Cloud providers. In this deliverable, we report our experience in the deployment of the distributed shuffle index in the high-performance EMC environment. The evaluation of the performance of the distributed shuffle index demonstrates its practical integrability and the benefits that can be obtained by the use of Cloud infrastructures supporting the integration of multiple Cloud providers. We also analyzed the use of distributed Cloud storage and All-or-Nothing-Transform encryption modes (AONT) to effectively provide security guarantees.

Parallel to this line of work, we have addressed the problem of analyzing Cloud services according to users' needs. In fact, while the availability of multiple Cloud providers and services can be leveraged for security, it also makes the Cloud market a complex and diversified place. This aspect was already noted in the context of Task 4.1, which investigated different characteristics of CSPs (identifying SLOs and SLAs) and metrics to assess their security services. While in Task 4.1 user requirements could be taken into consideration, they were assumed to be expressed through the same SLA template used by the CSP themselves. Task 4.2 instead focuses more on the user-side perspective of the problem, aiming to support users in formulating arbitrary requirements and preferences over the characteristics of Cloud providers and take them into account in CSP's selection. The ultimate goal is an expressive, powerful, and high-level language that users can adopt to easily express their requirements and preferences. Such requirements and preferences could then be considered to differentiate acceptable Cloud providers and services available in multi-cloud scenarios, providing users with solutions that best fit their needs. In this deliverable, we then also illustrate our solution for allowing users to formulate requirements and preferences on Cloud service characteristics, together with an approach to enforce them.

The remainder of this chapter is organized as follows. Section 2.1 presents a compact overview of the innovative results achieved within Task 4.2. Section 2.2 presents an integration of the distributed shuffle index with EMC ECS described in [BBB+17], discussing also its promising experimental results. Section 2.3 analyzes the use of distributed Cloud storage, observing how an AONT transformation could effectively provide security guarantees. Section 2.4 illustrates our approach for specifying and enforcing user requirements and preferences on the properties of Cloud services to evaluate their suitability with respect to the user's needs.

2.1 ESCUDO-CLOUD Innovation

This task produced several advancements over the state of the art.

- We proposed a technique that, building on the shuffle index approach studied in Work Package 2, randomly partitions data among three independent Cloud providers and ensures access confidentiality by dynamically re-allocating data at every access (D4.2). We analyzed different scenarios, and confirmed that distributed allocation and swapping provide good protection guarantees, even in presence of collusion among the providers.

- We presented the architectural and design issues addressed for the implementation of the distributed shuffle index integrated with different Cloud providers (D4.3). We compared the performance obtained on the local infrastructure, and then run another series of experiments for recording performance on the high-performance cooperative Cloud infrastructure supported by EMC.

- We deployed the distributed shuffle index in the high-performance EMC environment, and report our experience in this document. The evaluation of the performance of the distributed shuffle index demonstrates its practical integrability and the benefits that can be obtained by the use of Cloud infrastructures supporting the integration of multiple Cloud providers.

- We analyzed the use of distributed Cloud storage, and studied how an All-or-Nothing-Transform encryption mode can effectively provide security guarantees. We present our experience in this field in this document.

- We proposed a user-friendly and expressive approach for allowing users to specify requirements and preferences in the selection of the Cloud services to be adopted in a multi-provider scenario. Our approach supports users in moving in the diversified Cloud market, to choose the providers/services that are more compliant to their needs.

2.2 Distributed Shuffle Index: Analysis and Implementation in an Industrial Testbed

The problem of protecting data confidentiality has been widely addressed by the research community. Encryption is typically employed to protect data at-rest and data in-transit. However, by observing accesses, a Cloud provider could infer sensitive information about the user performing the access and the possibly sensitive content of the outsourced dataset. Recently, several approaches have been proposed for protecting access and pattern confidentiality (e.g., [BDF+17, DFP+11, DvDF+16, DFP+15, SvS+13, SS13, DFP+17]). Among them, the *distributed shuffle index* [DFP+17] is a B+-tree index structure that enables efficient key-based data retrieval, while guaranteeing content, access, and pattern confidentiality. It relies on the presence of three independent Cloud providers to improve the protection guarantees offered by the single-provider shuffle index [DFP+11]. The distributed shuffle index is based on the combined adoption of *data distribution* and *swapping* protection techniques. Data distribution consists in allocating the nodes composing the shuffle index to three different and independent Cloud providers. Each provider will then store and have visibility on accesses only on a portion of the index structure. Swapping consists in continuously changing the physical allocation of accessed data, which are moved to a different Cloud provider after each access. The combined adoption of data distribution and swapping guarantees protection of access confidentiality also in case of collusion among the Cloud providers. With this configuration, a Cloud provider may not even know that data have been distributed, and observes accesses to only a portion of the shuffle index, while assuming that it is

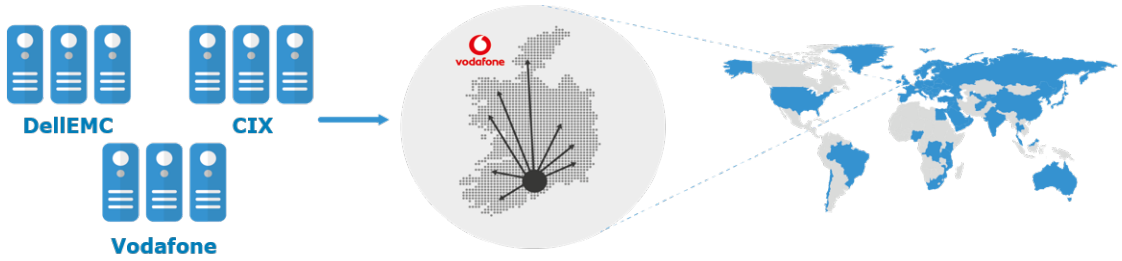

Figure 2.1: INFINITE testbed

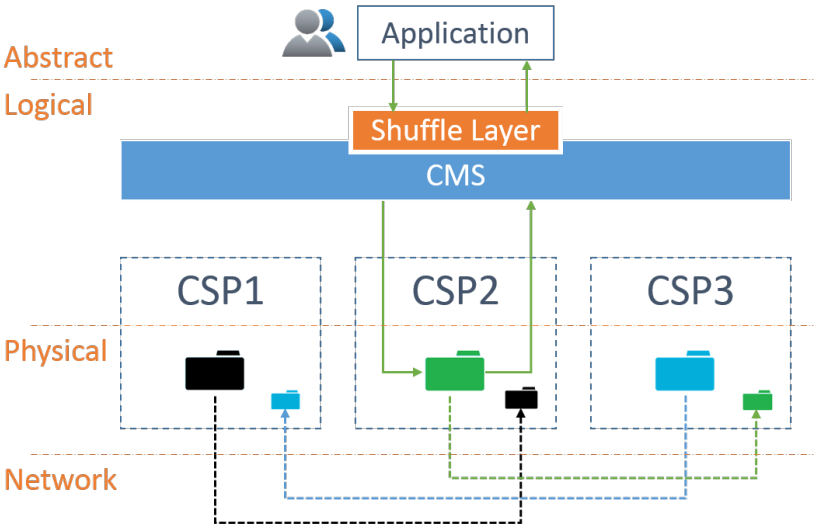

Figure 2.2: Shuffle index deployment model

observing the whole dataset.

2.2.1 Industrial Prototype of the Distributed Shuffle Index

The distributed shuffle index has been implemented in a prototype and deployed in EMC's INFINITE (INternational Future INdustrial Internet TEstbed [1]) platform. INFINITE, already introduced in [BdVF+17], is an initiative led by EMC, Vodafone Ireland and partners, and is an innovation platform built for the development of Industrial IoT products and solutions across a wide and diverse range of industries and sectors. The INFINITE testbed is composed of a full mobile network (2G to LTE) covering whole Ireland, and a Cloud infrastructure with a distributed data center architecture (Figure 2.1). In particular, the data center architecture connects three geographically diverse sites, i.e., EMC, Vodafone, and CIX data centers. The core network is an MPLS dark fiber ring offering 10 Gbps capacity.

Deployment model

Elastic Cloud Storage (ECS) is a turnkey software-defined, Cloud-scale, object storage platform selected as a target application for integration with the shuffle index. This environment for the prototype replicates a multi-Cloud data storage service. The prototype implementation is based on Python due to the wide availability of production-ready libraries that support the interaction with

[1] www.iotinfinite.org

	Plain encrypted index	*Distributed shuffle index*
ECS	$0.04210s$ $\sigma = 0.01322s$	$0.19674s$ $\sigma = 0.08075s$
Commercial Providers	$0.56777s$ $\sigma = 0.25588s$	$1.07609s$ $\sigma = 0.42817s$

Figure 2.3: Access times and their standard deviation σ

multiple Cloud providers. The deployment consists of four VMs (Figure 2.2). The shuffle index was deployed on a VM running a Content Management Service responsible for the distribution of data across ECS nodes. ECS nodes were deployed to the testbed in three installations configured on CentOS 7 VMs. ECS was set up as three single-node ECS configurations rather than one multi-node distributed ECS cluster. In this way, we simulate three separate and isolated Cloud providers that are not aware of each other. Once operational, each single-node configuration of ECS was migrated to a different physical location of the testbed.

2.2.2 Experimental results

We evaluate the impact on performance of the protection techniques adopted by a distributed shuffle index using two different Cloud providers, ECS and widely used public Cloud providers, referred in the following as "Commercial Provider". We omit their names as they are not relevant for the comparison between an infrastructure that offers high performance in the interconnectivity among systems controlled by independent entities and generic public Cloud providers. Our performance analysis utilizes two configurations, one for the ECS - INFINITE testbed deployment and one for the Commercial Providers. The data structure used for the experiments is a 2-level index with fan-out 27 and we performed 100 accesses over the index. Figure 2.3 shows a comparison between the distributed shuffle index and a plain encrypted index with the same static structure (i.e., a 2-level B+-tree with fan-out 27). The figure reports the average access time and the standard deviation σ.

Comparing the ECS performance with the Commercial Providers one, we can see one order of magnitude deterioration in performance. This is mainly due to the fact that the transfer rate was lower (100 Mbps), and the latency caused by geographical distance was higher than in the INFINITE testbed. This proves that an industrial high-speed closed area network would be the ideal application scenario for the distributed shuffle index, since the network latency is lessened and does not consist in a bottleneck any more. We note that in a slower network scenario, the distributed shuffle index still works well, but in this case the overhead introduced by the network is not negligible (and represents a bottleneck).

2.3 Distributed Cloud Storage

With Cloud technology, what was in the past managed autonomously now sees the involvement of servers often in an unknown location and immediately reachable wherever an Internet connection is present. The use of these Internet services today mostly assumes the presence of a Cloud Service Provider (CSP) managing the service. Yet, many users have an excess of computational, storage, and network capacity in the systems they own and would be interested in offering these resources to other users in exchange of a rent payment covering part of their costs. In the classical behavior

of markets, the existence of an infrastructure that supports the meeting of supply and demand for IT services would lead to a significant opportunity for the creation of economic value from the use of otherwise underutilized resources. Even if instances of cooperative applications are limited, today, to a few cases where there are legislative more than technical obstacles to the realization of a service by CSPs (e.g., P2P resource sharing supported by protocols like BitTorrent), the realization of services using the resources of end-users is one of the expected IT revolutions. This change of landscape is witnessed by the increasing attention of the research and development community toward the realization of *Distributed Cloud Storage* (DCS) services, characterized by the availability of multiple nodes that can be used to store resources in a distributed manner. In such services, individual resources are fragmented in *shards* allocated (with replication to provide availability guarantees) to different nodes. Access to a resource requires retrieving all its shards. The main characteristics of a DCS is the cooperative and dynamic structure formed by independent nodes (providing a multi-authority storage network) that can join the service and offer storage space, typically in exchange of some reward.

However, if security concerns and perception of (or actual) *loss of control* have been an issue and slowing factor for centralized Clouds, they are even more so for a decentralized Cloud storage, where the dynamic and independent nature of the network may hint to a further decrease of control of the owners on where and how their resources are managed. Client-side encryption typically assumed in DCSs provides only a basic form of protection, leaving resources exposed to potentially malicious nodes. Even if fragmented and distributed in the network, resources remain exposed to malicious attacks (e.g., tampering or brute-force attempts to bypass the natural cryptographic protection). Also, resources remain vulnerable in case of exposure of the encryption key or in case of malicious nodes non deleting resources upon the owner's request (i.e., secure deletion is not guaranteed) or colluding to reconstruct a resource in its entirely.

On the positive side, however, we note that the decentralized nature of DCS systems also increases the reliability of the service, as the involvement of a collection of independent parties reduces the risk that a single malfunction can limit the accessibility to the stored resources. In addition to this, the independent structure characterizing DCS systems - if coupled with effective resource protection and careful allocation to nodes in the network - makes them promising for actually strengthening security guarantees for owners relying on the decentralized network for storing their data.

Given the objective of Task 4.2 to formulate techniques that leverage the presence of multiple providers to enhance security, we also identified as an interesting approach the client-side application of an *All-Or-Nothing-Transform* (AONT) encryption mode transforming resources for their external storage. This mode requires the use of an encryption key. The encryption driven by the key represents the primary protection, and the AONT complements it and further increases security. An AONT-encryption mode transforms a plaintext resource (original content in whatever form) into a ciphertext, with the property that the complete result of the transformation is required to obtain back the original plaintext. AONT guarantees in fact complete interdependence (*mixing*) among the bits of the encrypted resource in such a way that the unavailability of a portion of the encrypted resource prevents the reconstruction of any portion of the original plaintext. A party having access to a portion of the encrypted resource (but not to the encrypted resource in its entirety), if knowing the cryptographic key, will not be able to reconstruct any portion of the resource. Consequently, if not knowing the cryptographic key, the party will not be able to perform brute-force attacks for guessing such a key, as any key (even the correct one) will be ineffective if not applied to the complete resource.

In a DCS context, the use of AONT guarantees protection to the individual shards composing the resource, and therefore to the resource itself (in its entirety as well as any of its portion).

2.4 Supporting User Constraints and Preferences in Cloud Adoption

We now illustrate our approach for allowing a user to specify constraints and preferences that a Cloud service should satisfy to be considered for outsourcing. The enforcement of such requirements allows the user to evaluate the multiple offers available in the Cloud market, and to choose the service(s) that matches her needs. After a brief discussion on basic concepts and rationale (Section 2.4.1), we illustrate our approach for specifying user constraints (Sections 2.4.2 and 2.4.3) and for enforcing them (Sections 2.4.4 and 2.4.5). We then discuss how users can also specify preferences that can make one service more appealing than another (Section 2.4.6).

2.4.1 Basic Concepts and Rationale

We consider a scenario characterized by a user who wants to outsource her data and applications to a Cloud service provider (or a set thereof), and is interested in comparing the products available on the Cloud market to properly decide which is the most suitable to her needs.

We assume the existence of a set $\mathbb{S} = \{S_1, \ldots, S_n\}$ of candidate Cloud services, offered by the providers on the market. Each candidate service $S \in \mathbb{S}$ is characterized by a set of (functional and non-functional) *properties*, such as the guaranteed availability and the enforced security mechanisms, ensured during the service provision. We model such properties as *attributes* $\mathbb{A} = \{A_1, \ldots, A_m\}$, defined in the order over domains $Dom(A_1), \ldots, Dom(A_m)$ and taken from a common ontology shared between the providers and the user [DLP16]. For simplicity, we assume attributes to have finite cardinality (i.e., $Dom(A) = \{v_1, \ldots, v_l\}$). Each service S is characterized by (and differs from the others because of) the set of values assigned to (a subset of) the attributes in $\mathbb{A}$. We assume here that each attribute can take only one value in its domain. Note that a Cloud provider might also not provide any guarantee on the value it can offer with respect to a specific property. For instance, a provider could decide not to specify how many backup copies of data it takes. This does not mean that the provider does not backup users' data, but that it cannot/does not wish to give any formal guarantee on that. Figure 2.4 illustrates an example of a set $\mathbb{A}$ of attributes of interest for the user together with their domains, extended with the special value $*$ to indicate no value. If not explicitly stated otherwise, for simplicity, when referring to the domain $Dom(A)$ of an attribute A, we assume $*$ to be included in $Dom(A)$. In the figure and in the remainder of this section, attribute `loc` models the physical location of the servers used by a service, `encr` the adopted encryption algorithm, `rep` the number of maintained replicas of the outsourced data, `avail` the guaranteed availability level, `pTest` the authority running penetration testing (which could performed either internally or by two different authorities), `cert` the security certification issued to the service, and `audit` the frequency of security auditing.

We denote the properties characterizing a Cloud service S as a vector pv_S, with a cell for each attribute A in $\mathbb{A}$, as formally defined in the following.

Definition 1 (Property vector)**.** *Let S be a Cloud service and $\mathbb{A} = \{A_1, \ldots, A_m\}$ a set of attributes defined, in the order, over domains $Dom(A_1), \ldots, Dom(A_m)$. The* property vector *of service S is a vector $\mathsf{pv}_S[A_1, \ldots, A_m]$, where $\mathsf{pv}_S[A_i] \in Dom(A_i)$ $(i = 1, \ldots, m)$ is the value S ensures for A_i.*

The property vector represents the value, for each attribute of interest for the user, guaranteed

Attribute A	**Domain** $Dom(A)$
`loc`	{US, EU, Japan, $*$}
`encr`	{AES, 3DES, DES, $*$}
`rep`	{1, 2, 3, 4, $*$}
`avail`	{95%, 96%, 97%, 98%, 99%, 99.2%, 99.5%, $*$}
`pTest`	{int, authA, authB, $*$}
`cert`	{certA, certB, certC, certD, $*$}
`audit`	{3M, 6M, 1Y, $*$}

Figure 2.4: An example of a set $\mathbb{A}$ of attributes with their domains

	`loc`	`encr`	`rep`	`avail`	`pTest`	`cert`	`audit`
S_1	Japan	DES	4	97%	int	certC	$*$
S_2	US	3DES	3	98%	int	certB	1Y
S_3	US	AES	2	99.2%	authB	certC	$*$
S_4	EU	3DES	3	99.5%	authB	certB	$*$
S_5	US	AES	3	99.2%	authA	certC	$*$
S_6	EU	AES	3	99.5%	authA	certA	$*$
S_7	US	AES	3	99.5%	authB	certC	$*$

Figure 2.5: Tabular representation of an example of property vectors pv_S for a set $\mathbb{S} = \{S_1, \ldots, S_7\}$ of Cloud services

by a Cloud service. In other words, it represents in a compact way the set of properties characterizing a service S that are of interest to the user. Entry $\mathsf{pv}_S[A_i]$ in the property vector represents the guarantee assured by the service for attribute A, if it is equal to a value $v \in Dom(A_i) \setminus \{*\}$, or that not guarantee is given, otherwise. Figure 2.5 illustrates in tabular form an example of a set of property vectors pv_S for a set $\mathbb{S} = \{S_1, \ldots, S_7\}$ of Cloud services. For instance, $\mathsf{pv}_{S_1} =$ [Japan, DES, 4, 97%, int, certC, $*$] meaning, for instance, that service S uses servers physically located in Japan ($\mathsf{pv}_{S_1}[\mathtt{loc}]$ =Japan) and does not provide guarantees on the security auditing frequency ($\mathsf{pv}_{S_1}[\mathtt{audit}] = *$).

When moving to the Cloud, users typically have requirements that a candidate Cloud provider should satisfy to be considered for outsourcing. These can be either *hard* requirements, whose satisfaction by a service S is mandatory for S to be considered for outsourcing, as well as *soft* requirements, whose satisfaction by a service S_i makes S_i more appealing than another service S_j that does not satisfy them (despite both S_i and S_j being possible candidates w.r.t. hard requirements). Hard requirements naturally translate to *constraints* that identify the values that a certain property of interest can or cannot assume. For instance, a user might want to consider only services ensuring a minimum availability level of 99%: a service ensuring a lower level, or not ensuring any level at all, cannot intuitively be a correct choice for the user. Soft constraints, on the other hand, naturally translate to *preferences* that the user might have over the (acceptable) values that a property can assume. For instance, a user might consider acceptable both the AES and 3DES ciphers, while preferring AES to 3DES for the security guarantees it offers. Our goal is to propose a simple yet expressive framework that can be adopted by a user to specify her constraints and preferences. As for user constraints, our model builds upon two basic building blocks (referred to as *base constraints* and illustrated in Section 2.4.2) that can be used by the user to specify, for a given attribute, which are the values that can or cannot be assumed by a candidate service to be considered for outsourcing. To enrich the expressive power provided to the user, these building blocks can possibly be enforced as they are, or combined in more complex Boolean formulas (referred to

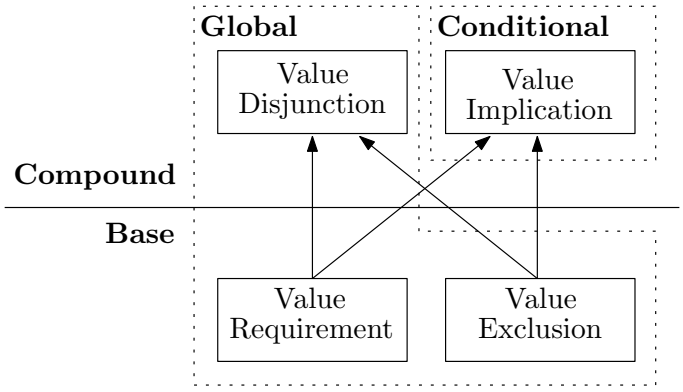

Figure 2.6: Classification of the constraints in our model

as *compound constraints* and illustrated in Sections 2.4.2 and 2.4.3) to allow the user to require alternatives, disjunctions, conjunctions, and implications among them. A *constraint policy*, that is, a set of base and compound constraints, can therefore be formulated by a user to specify those requirements that must be satisfied by a candidate service to be considered for outsourcing. Given a set $\mathbb{S}$ of candidate services, therefore, the goal is that of using the constraint policy to determine the set $\mathbb{S}^v \subseteq \mathbb{S}$ of services that satisfy the needs of the user, and to further refine $\mathbb{S}^v$ by enforcing user preferences (i.e., her soft requirements).

2.4.2 Global Constraints

Figure 2.6 graphically illustrates our constraints: base constraints (i.e., value requirements and exclusions as will be clarified in this section) can be used to formulate compound constraints (value disjunctions and implications), and these constraints can also be classified in global versus conditional. In a nutshell, global constraints apply to candidate services, while conditional constraints apply only to all candidate services that satisfy user-defined conditions. We will illustrate more details on the global constraints in this section, and the conditional constraints in Section 2.4.3.

Given a set $\mathbb{S}$ of candidate Cloud services, a global constraint applies to all services in $\mathbb{S}$, and can be used by the user to identify the sets of values that are considered acceptable (or, similarly, not acceptable) for some properties of interest (attributes in $\mathbb{A}$ in our modeling) characterizing the services in $\mathbb{S}$.

The most straightforward kind of such a constraint aims at restricting the values that each attribute, individually taken, can assume: for instance, to ensure adequate confidentiality against unauthorized parties, a user might wish to consider only services encrypting data in storage with the AES cipher (i.e., in our model, assuming value AES for attribute `encr` in their property vectors). Moreover, since Cloud services feature several different properties, alternatives among such constraints might be specified by the user: for instance, to ensure adequate security, a user might wish to consider only services that adopt AES cipher *or* comply with a specific security certification. We will first introduce base constraints operating on single attributes, to then highlight how they can be possibly combined to enrich the expressive power of users.

Base constraints. A *base constraint* is a global constraint that operates over a single attribute $A \in \mathbb{A}$ restricting the values that A can assume in the service provision. In our scenario, it restricts the original domain $Dom(A)$ to a subset of $Dom(A)$ including only the values that are acceptable to the user. Such a constraint can be formulated by either explicitly listing the values that A can

R_1: ⟨loc IN {US, EU}⟩
R_2: ⟨repl IN {2,3,4}⟩

E_1: ⟨encr NOT IN {DES}⟩
E_2: ⟨avail NOT IN {95%,96%}⟩

D_1: ⟨pTest IN {authA, authB}⟩∨ ⟨cert IN {certA, certB}⟩
D_2: ⟨loc IN {EU}⟩∨ ⟨cert IN {certC}⟩

I_1: ⟨loc IN {US}⟩∧⟨encr IN{3DES}⟩⟹⟨auditIN {3M, 6M}⟩∨
⟨pTest IN {certA}⟩
I_2: ⟨loc IN {EU}⟩⟹⟨cert NOT IN {certC}⟩

Figure 2.7: An example of user constraints

assume (*value requirement*) or, equivalently, specifying the values that A cannot assume (*value exclusion*), as follows.

Definition 2 (Value requirement)**.** *Given a set $\mathbb{A}$ of attributes and an attribute $A \in \mathbb{A}$ with domain $Dom(A)$, a* value requirement *R over A is an expression of the form $R = \langle A \text{ IN } \{v_1, \ldots, v_h\}\rangle$, with $v_1, \ldots, v_h \in Dom(A)$.*

The intuitive semantics of a value requirement $R = \langle A \text{ IN } \{v_1, \ldots, v_h\}\rangle$ bounds A to assume a value among $v_1, \ldots, v_h$. Indeed, a value requirement can also include a single accepted value (i.e., $R = \langle A \text{ IN } \{v_i\}\rangle$, with $v_i \in Dom(A)$). Figure 2.7 includes two value requirements R_1 and R_2 over the set $\mathbb{A}$ of attributes in Figure 2.4, identifying the sets of values that attributes `loc` and `repl` can assume (i.e., values US and EU for `loc` by R_1, and 2, 3, and 4 for `repl` by R_2).

Value exclusions are complementary to value requirements, and model the fact that the user requires that an attribute $A \in \mathbb{A}$ cannot assume a specific value (or a set thereof) considered forbidden. Value exclusions are formally defined as follows.

Definition 3 (Value exclusion)**.** *Given a set $\mathbb{A}$ of attributes and an attribute $A \in \mathbb{A}$ with domain $Dom(A)$, a* value exclusion *E over A is an expression of the form $E = \langle A \text{ NOT IN } \{v_1, \ldots, v_h\}\rangle$, with $v_1, \ldots, v_h \in Dom(A)$.*

The semantics of a value exclusion $E = \langle A \text{ NOT IN } \{v_1, \ldots, v_h\}\rangle$ bounds A *not* to assume a value among $v_1, \ldots, v_h$. Indeed, like for requirements, a value exclusion can also include a single forbidden value (i.e., $E = A \text{ NOT IN } \{v_i\}$, with $v_i \in Dom(A)$). Figure 2.7 includes two value exclusions E_1 and E_2 over the set $\mathbb{A}$ of attributes in Figure 2.4, identifying the sets of values that attributes `encr` and `avail` cannot assume (i.e., value DES for `encr` by E_1, values 95% and 96% for `avail` by E_2, and the $*$ value for both attributes by E_1 and E_2).

The practical effect of the enforcement of a value requirement or a value exclusion over an attribute A is to redefine the domain $Dom(A)$ into a subset of acceptable values. To this end, requirements and exclusions are complementary and given an attribute A with domain $Dom(A)$, the same set of acceptable values could be obtained by enforcing both a value requirement or a value exclusion over A. To illustrate, considering the domain {EU, US, Japan, $*$} of attribute `loc` in Figure 2.4, the same set {EU, US} of acceptable values induced by requirement R_1 in Figure 2.7 could also be obtained through the definition of an exclusion of the form ⟨loc NOT IN {Japan, $*$}⟩. Whether to specify requirements or exclusions can be freely chosen by a user, depending on her perceived convenience. Also, we assume that for each attribute $A \in \mathbb{A}$, at most one among either a value requirement, or a value exclusion, can be defined. To illustrate, consider the case in which

a combination of value requirements and exclusions be defined over the same attribute A. Indeed, the only situation which would make sense in our scenario is when all such constraints induce the same set of acceptable values, hence making all the constraints but one redundant.

Given an attribute A with domain $Dom(A)$, and a base constraint defined over it, we define the set of acceptable values specified by the constraint as its *constrained domain* $Dom^+(A) \subseteq Dom(A)$. More precisely, given a value requirement $R = \langle A \text{ IN } \{v_1,\ldots,v_h\}\rangle$, the domain $Dom^+(A)$ constrained by R is defined as $Dom^+(A) = \{v_1,\ldots,v_h\}$. Similarly, given a value exclusion $E = \langle A \text{ NOT IN } \{v_1,\ldots,v_h\}\rangle$, the domain $Dom^+(A)$ constrained by E is defined as $Dom^+(A) = Dom(A) \setminus \{*, v_1,\ldots,v_h\}$. For instance, with reference to attributes `loc` and `encr` in Figure 2.4 and the base requirements in Figure 2.7, $Dom^+(\texttt{loc}) = \{\text{US, EU}\}$, and $Dom^+(\texttt{encr}) = \{\text{3DES, AES}\}$. Intuitively, a Cloud service can be considered for outsourcing only if the values assumed by the property modeled by A is included in $Dom^+(A)$. To illustrate, consider the Cloud services in Figure 2.5: according to the constraints in Figure 2.7, S_1 could not be adopted for outsourcing since it assumes value Japan for attribute `loc` (i.e., not satisfying R_1) and value DES for attribute `encr` (i.e., not satisfying E_1). In this regard, as will be illustrated in detail in Section 2.4.5, all services assuming for attribute A values not included in $Dom^+(A)$ could be directly excluded as they do not satisfy a base requirement of the user. Note that the automatic exclusion of value $*$ from domains constrained by exclusions naturally follows the semantics of value $*$, modeling the fact that no guarantee is given for the value assumed by a service for an attribute: if $*$ were included in constrained domains, there would be no guarantee that S does not assume a value among those explicitly forbidden by the exclusion.

Compound constraints. Having illustrated our base constraints, we can now focus on how they can be combined to the aim of augmenting the expressive power of the user. Note that combining multiple base constraints over a single attribute would not make sense in our scenario (as requirements and exclusions are complementary as illustrated above), and therefore in this section we refer to combining base constraints over different attributes.

Natural combinations among base requirements over different attributes translate, in our scenario, to Boolean conjunctions and disjunctions among them. However, let us note that conjunctions of the form $\langle A_i \text{ OP } \{\ldots\}\rangle \wedge \ldots \wedge \langle A_j \text{ OP } \{\ldots\}\rangle$ are naturally captured by our model, without the need of a specific compound constraint: since each base constraint is defined on a different attribute, such a conjunction is satisfied iff each of the base constraints is satisfied as is. The semantics of a conjunction among k base constraints defined over k different attributes is then captured by the specification of the k base constraints themselves. To illustrate, with reference to the base constraints in Figure 2.7, enforcing both R_1 and R_2 would entail the same result than that of enforcing the Boolean conjunction $R_1 \wedge R_2$ (i.e., a service satisfying both R_1 and R_2 would certainly satisfy also the conjunction among them). On the contrary, disjunctions among base requirements need a specific formulation, and to this aim we introduce the concept of *value disjunction*, formally defined as follows.

Definition 4 (Value disjunction)**.** *Given a set* $\mathbb{A}$ *of attributes, a* value disjunction D *over a subset* $\{A_1,\ldots,A_w\} \subseteq \mathbb{A}, |\{A_1,\ldots,A_w\}| \geq 2$*, is an expression of the form* $D = \bigvee_{i=1}^{w}(A_i \text{ OP } \{v_{i_1},\ldots,v_{i_h}\})$*, with* $\text{OP} \in \{\text{IN}, \text{NOT IN}\}$ *and* $v_{i_1},\ldots,v_{i_h} \in Dom(A_i)$.

According to Definition 4, a value disjunction is a Boolean formula defined over a set of value requirements and/or exclusions, linked by the Boolean disjunction operator. To this end, a value disjunction D can be used to specify a set of base constraints that are somehow considered equivalent (or, in other words, mutually exchangeable), and such that the satisfaction of at least

one constraint by a service S is sufficient to consider D as satisfied by S. Figure 2.7 includes two value disjunctions D_1 and D_2 over subsets of the set $\mathbb{A}$ of attributes in Figure 2.4. For instance, disjunction D_1, defined over attributes `pTest` and `cert`, requires that `pTest` assumes a value included in {authA, authB}, or that `cert` assumes a value included in {certA, certB}.

Considering that a value disjunction is built upon base constraints, and recalling the semantics of such base constraints (i.e., restricting the domain $Dom(A)$ to a constrained domain $Dom^{+}(A) \subseteq Dom(A)$), given a value disjunction D over a set $\{A_1,\ldots,A_w\} \subseteq \mathbb{A}$ of attributes, a Cloud service S can be considered for outsourcing iff, for *at least* one attribute A among $\{A_1,\ldots,A_w\}$, it assumes a value acceptable according to the base constraint operating on A. For instance, consider value disjunction D_1 in Figure 2.7: a service can be considered for outsourcing only if it assumes one value among authA and authB for `pTest`, or one value among certA and certB for `audit`. With reference to the set of services in Figure 2.5, all services but S_1 satisfy D_1.

As a last remark, we underline that value disjunctions might also be adopted to specify mutually exclusive alternatives among base constraints (e.g., compound constraints of the form *at most one among a set of k base requirements must be satisfied*). To this aim, it is however necessary to combine value disjunctions with conditional constraints, illustrated in the next section.

2.4.3 Conditional Constraints

Conditional constraints are compound constraints (i.e., like value disjunctions) that are used to identify the values that can or cannot be assumed by a set of properties of a Cloud service in $\mathbb{S}$, like the global constraints we have illustrated so far. They differ from global constraints since, rather than applying to *all* services in $\mathbb{S}$, they might only apply to a subset of services in $\mathbb{S}$, that is, only to those services that satisfy some user-defined conditions. Building on our base constraints (i.e., value requirements and exclusions), conditional constraints can be used by the user to state that, for example, a certain encryption algorithm (e.g., 3DES) can be accepted only with a key of a certain minimum length (e.g., 112 bits). Let us recall that this requirement does not state that the user is interested in a service in $\mathbb{S}$ providing 3DES with a minimum key of 112 bits (which could be easily defined by specifying two different value requirements over the attributes modeling these properties), but that *if* a service S provides 3DES, *then* S can be considered for outsourcing *only if* the keylength is at least 112 bits long.

To this aim, we introduce the concept of *value implication*, formally defined as follows.

Definition 5 (Value implication)**.** *Given a set $\mathbb{A}$ of attributes and two subsets $\mathbb{A}', \mathbb{A}'' \subset \mathbb{A}$ such that $\mathbb{A}' \cap \mathbb{A}'' = \emptyset$, a* value implication *$I$ over $\mathbb{A}'$ and $\mathbb{A}''$ is an expression of the form $I = \bigwedge_{A_i \in \mathbb{A}'}(A_i \text{ OP } \{v_{i_1},\ldots,v_{i_h}\}) \Longrightarrow \bigvee_{A_j \in \mathbb{A}''}(A_j \text{ OP } \{v_{j_1},\ldots,v_{j_h}\})$, with $v_{i_1},\ldots,v_{i_h} \in Dom(A_i)$, $v_{j_1},\ldots,v_{j_h} \in Dom(A_j)$, and $\text{OP} \in \{\text{IN}, \text{NOT IN}\}$.*

According to Definition 5, a value implication I is a formula over two disjoint sets of attributes linked by the $\Longrightarrow$ symbol. We refer to the left hand side of an implication I as the *head* of I (denoted $I.h$), and to the right hand side of I as the *tail* of I (denoted $I.t$). Given an implication I, the head of I is represented by a Boolean conjunction of base constraints, while its tail is basically a value disjunction (Definition 4). Only those Cloud services that satisfy the head of I (i.e., such that for all attributes A in $\mathbb{A}'$ they assume a value included in $Dom^{+}(A)$ as restricted by the base constraint over A in $I.h$) are affected by I. Figure 2.7 includes two value implications I_1 and I_2 over pairs of subsets of the set $\mathbb{A}$ of attributes in Figure 2.4. For instance, value implication I_1 defined over $\mathbb{A}' = \{$`loc`,`encr`$\}$ and $\mathbb{A}'' = \{$`audit`,`pTest`$\}$ applies only to services taking values US and

3DES for attributes `loc` and `encr`, respectively, and requires them to assume values 6M or 3M for attribute `audit`, and authA for `pTest`. For instance, with reference to the candidate services in Figure 2.5, service S_2 does not satisfy I_1 since, while assuming values satisfying $I_1.h$ (i.e., US and 3DES), it does not assume values that also satisfy $I_1.t$.

It is worth underlining why a value implication connects constraints in its head through the Boolean conjunction operator, and the constraints in its tail through the Boolean disjunction operator (i.e., we do not consider implications with disjunctions in their heads, or with conjunctions in their tails). First, an implication I including in its tail a conjunction among a set of k base constraints can be easily expressed as a set of k implications with the same head as I, and one among the k base conditions in the tail. A service satisfying I will then certainly satisfy the entire set of k 'simplified' implications. For instance, a service satisfying a constraint of the form $\langle$`loc` IN $\{$EU$\}\rangle \Longrightarrow \langle$`cert` NOT IN $\{$certC$\}\rangle \wedge \langle$`audit` NOT IN $\{$6M$\}\rangle$indeed satisfies also both the constraints $\langle$`loc` IN $\{$EU$\}\rangle \Longrightarrow \langle$`cert` NOT IN $\{$certC$\}\rangle$ and $\langle$`loc` IN $\{$EU$\}\rangle \Longrightarrow \langle$`audit` NOT IN $\{$6M$\}\rangle$. Similarly, an implication I including in its head a disjunction among a set of k base constraints can be easily translated in a set of k implications with the same tail as I, and one among the k constraints in the head. A service satisfying I will then certainly satisfy at least one among the k 'simplified' implications. For instance, a service satisfying a constraint of the form $\langle$`loc` IN $\{$EU$\}\rangle \vee \langle$`avail` IN$\{99\}\rangle \Longrightarrow \langle$`cert` NOT IN $\{$certC$\}\rangle$would certainly satisfy at least one among the constraints $\langle$`loc` IN $\{$EU$\}\rangle\rangle \Longrightarrow \langle$`cert` NOT IN $\{$certC$\}\rangle$and $\langle$`avail` IN$\{99\}\rangle \Longrightarrow \langle$`cert` NOT IN $\{$certC$\}\rangle$.

As mentioned in Section 2.4.2, conditional constraints can also be employed together with a value disjunction D to model mutually exclusive alternatives among the base constraints in D. To illustrate, consider the value disjunction D_2 and the value implication I_2 in Figure 2.7. It is immediate to see that requiring the enforcement of both constraints can be seen as a translation of the disjunction D_2 (naturally a Boolean disjunction among two basic constraints, and therefore satisfied if at least one among them is satisfied) into an exclusive disjunction, requiring that a candidate service must assume *either* value EU for `loc`, *or* value certC for `cert`, but not both.

2.4.4 Constraint policy

The global and conditional constraints illustrated above (i.e., value requirements, exclusions, disjunctions, and implications) can be easily specified by a user to formulate the requirements that a Cloud service S must satisfy to be considered by the user for outsourcing. We refer to the set of constraints specified by the user as her *constraint policy*. More precisely, a constraint policy $\mathscr{C}^{\mathbb{A}}$ over a set $\mathbb{A}$ of attributes is a tuple $\mathscr{C}^{\mathbb{A}} : \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ with $\mathbb{R}$ a set of value requirements over attributes in $\mathbb{A}$ (Definition 2), $\mathbb{E}$ a set of value exclusions over attributes in $\mathbb{A}$ (Definition 3), $\mathbb{D}$ a set of value disjunctions over subsets of $\mathbb{A}$ (Definition 4), and $\mathbb{I}$ a set of value implications over pairs of subsets of $\mathbb{A}$ (Definition 5). For instance, the set of constraints in Figure 2.7 is an example of a constraint policy over the attributes in Figure 2.4, where $\mathbb{R} = \{R_1, R_2\}$, $\mathbb{E} = \{E_1, E_2\}$, $\mathbb{D} = \{D_1, D_2\}$, and $\mathbb{I} = \{I_1, I_2\}$.

In the remainder of this document, we will assume constraint policies to be *well-formed*, that is, including constraints that are not in conflict. A simple example of two constraints that are in conflict is represented by a value requirement $R = A$ IN $\{v\}$ and a value exclusion $E = A$ NOT IN $\{v\}$, whose joint enforcement is clearly not possible as any service S can satisfy either R or E, but not both. More generally, as illustrated in Section 2.4.2, for each attribute $A \in \mathbb{A}$, at most one among either a value requirement or a value exclusion can be defined. Before illustrating the

$$Dom^{+}(A) = \begin{cases} \{v_1, \ldots, v_h\}, & \text{if } \exists R \in \mathbb{R} \text{ s.t. } R = \langle A \text{ IN } \{v_1, \ldots, v_h\} \rangle \\ Dom(A) \setminus \{*, v_1, \ldots, v_h\}, & \text{if } \exists E \in \mathbb{E} \text{ s.t. } E = \langle A \text{ NOT IN } \{v_1, \ldots, v_h\} \rangle \\ Dom(A), & \text{otherwise} \end{cases}$$

Figure 2.8: Domain of attribute $A \in \mathbb{A}$ as constrained by a policy $\mathscr{C}^{\mathbb{A}} : \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$

conditions that a constraint policy must satisfy to be well-formed, let us formally define how the constrained domains of the attributes in $\mathbb{A}$ are built according to a constraint policy. Figure 2.8 illustrates how, given a constraint policy $\mathscr{C}^{\mathbb{A}}$ and an attribute $A \in \mathbb{A}$ with domain $Dom(A)$, the domain $Dom^{+}(A)$ constrained by $\mathscr{C}^{\mathbb{A}}$ is defined. As shown in the figure, $Dom^{+}(A)$ will include the values constrained by the requirement (or exclusion) defined on it, if any; otherwise it will be equal to $Dom(A)$. To illustrate, consider the attributes $\mathbb{A}$ and domains in Figure 2.4 and the constraint policy illustrated in Figure 2.7. The constrained domain of attribute `loc` is defined as $Dom^{+}(\texttt{loc}) = \{\text{EU, US}\}$ due to the fact that $\mathscr{C}^{\mathbb{A}}$ includes requirement R_1 over `loc`. On the contrary, the constrained domain of attribute `pTest` will be defined as $Dom^{+}(\texttt{pTest}) = Dom(\texttt{pTest})$, since $\mathscr{C}^{\mathbb{A}}$ includes neither a requirement nor an exclusion over `pTest`.

Given a policy $\mathscr{C}^{\mathbb{A}} : \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ and an attribute $A \in \mathbb{A}$, let us denote with notation $\mathbb{R}(A)$ and $\mathbb{E}(A)$ the set of value requirements and disjunctions in $\mathscr{C}^{\mathbb{A}}$ defined over A. Let us also denote with notation $\mathbb{D}(A)$ and $\mathbb{I}(A)$ the set of value disjunctions and implications in $\mathscr{C}^{\mathbb{A}}$ including a base constraint over A. Formally, a well-formed constraint policy is defined as follows.

Definition 6 (Well-formed constraint policy)**.** *Given a set $\mathbb{A}$ of attributes, a constraint policy $\mathscr{C}^{\mathbb{A}} : \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ over $\mathbb{A}$ is said to be* well-formed *iff the following conditions hold $\forall A \in \mathbb{A}$:*

1. $|\mathbb{R}(A)| + |\mathbb{E}(A)| \leq 1$
2. $\forall D \in \mathbb{D}(A) : Dom(A)|D \subseteq Dom^{+}(A)$
3. $\forall I \in \mathbb{I}(A) : Dom(A)|I \subseteq Dom^{+}(A)$

with $Dom(A)|D$ ($Dom(A)|I$, resp.) the set of values in $Dom(A)$ restricted by the base requirement in D (in I, resp.) defined over A.

Condition 1 ensures that, for each attribute A, no more than one among a requirement and an exclusion can be included in a well-formed policy. Conditions 2 and 3 ensure that, for each attribute A, all value disjunctions (and value implications) defined over subsets of $\mathbb{A}$ including A are compliant with the (constrained) domain of A (i.e., $Dom(A)$ if no explicit value requirement / exclusion in the policy has restricted it, $Dom^{+}(A)$ otherwise, see Figure 2.8). If these two latter conditions were not satisfied, then at least one among a value disjunction / implication would be defined over values not acceptable by requirements / exclusions in $\mathbb{R}$ and $\mathbb{E}$.

It is easy to see that the constraint policy $\mathscr{C}^{\mathbb{A}}$ illustrated in Figure 2.7 and defined over the set $\mathbb{A}$ of attributes in Figure 2.4 is well-formed according to Definition 6. In fact, it includes at most one among a value requirement or a constraint for each attribute in $\mathbb{A}$, satisfying Condition 1, and no base constraint included in sets $\mathbb{D}$ and $\mathbb{I}$ requires a value for an attribute A that is not included in $Dom^{+}(A)$, satisfying Conditions 2 and 3.

2.4.5 Valid Service

Given a set $\mathbb{A}$ of attributes modeling the characteristics of a set $\mathbb{S}$ of Cloud services, and a well-formed constraint policy $\mathscr{C}^{\mathbb{A}}$ over $\mathbb{A}$, we are now ready to formally define when a service $S \in \mathbb{S}$ satisfies $\mathscr{C}^{\mathbb{A}}$, being thus considerable for outsourcing. A service S satisfying a constraint policy $\mathscr{C}^{\mathbb{A}}$ is said to be *valid* w.r.t. $\mathscr{C}^{\mathbb{A}}$ and is formally defined as follows.

Definition 7 (Valid service)**.** *Given a set $\mathbb{A}$ of attributes, a set $\mathbb{S}$ of Cloud services, and a well-formed constraint policy $\mathscr{C}^{\mathbb{A}} : \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ over $\mathbb{A}$, a service $S \in \mathbb{S}$ with property vector pv_S is said to be* valid *w.r.t. $\mathscr{C}^{\mathbb{A}}$ iff the following conditions hold:*

1. $\forall A \in \mathbb{A}, \mathsf{pv}_S[A] \in Dom^+(A)$*;*
2. $\forall D \in \mathbb{D}$ *s.t.* $D = \bigvee_{i=1}^{w}(A_i \text{ OP } \{v_{i_1}, \ldots, v_{i_h}\}), \exists A \in \{A_1, \ldots, A_w\}$ *s.t.* $\mathsf{pv}_S[A] \in Dom(A)|D$*;*
3. $\forall I \in \mathbb{I}$ *s.t.* $I = \bigwedge_{A_i \in \mathbb{A}'}(A_i \text{ OP } \{v_{i_1}, \ldots, v_{i_h}\}) \implies \bigvee_{A_j \in \mathbb{A}''}(A_j \text{ OP } \{v_{j_1}, \ldots, v_{j_h}\})$*, if* $\forall A_i \in \mathbb{A}', \mathsf{pv}_S[A_i] \in Dom(A_i)|I$ *then* $\exists A_j \in \mathbb{A}''$ *s.t.* $\mathsf{pv}_S[A_j] \in Dom(A_j)|I$*.*

A service S valid w.r.t. a well-formed policy $\mathscr{C}^{\mathbb{A}}$ is guaranteed to satisfy the constraints included in $\mathscr{C}^{\mathbb{A}}$. Condition 1 ensures that, for all attributes A over which a value requirement or exclusion has been defined, S assumes a value deemed acceptable by such constraint. This guarantees that all value requirements and all value exclusions in $\mathscr{C}^{\mathbb{A}}$ be satisfied by S. Condition 2 ensures that each value disjunction in $\mathscr{C}^{\mathbb{A}}$ be satisfied by S. This translates into ensuring that, given a value disjunction D defined over a set $\{A_1, \ldots A_w\} \subseteq \mathbb{A}$ of attributes (and therefore composed of w requirements/exclusions over w attributes), S assumes for at least one of such w attributes a value deemed acceptable by the requirement/exclusion defined over it. Similarly, Condition 3 ensures that each value implication in $\mathscr{C}^{\mathbb{A}}$ be satisfied by S. This translates into ensuring that, given a value implication I defined over two sets $\mathbb{A}', \mathbb{A}'' \subset \mathbb{A}$ of attributes (and therefore composed of $|\mathbb{A}'|$ requirements/exclusions in the head and of $|\mathbb{A}''|$ requirements/exclusions in the tail), *if* the head of I is satisfied by S (i.e., all attributes in $\mathbb{A}'$ assume the values deemed acceptable by the conditions over them), *then* also its tail be satisfied by S. To illustrate, consider the set $\mathbb{A}$ of attributes in Figure 2.4, the constraint policy $\mathscr{C}^{\mathbb{A}}$ over $\mathbb{A}$ in Figure 2.7, and the set $\mathbb{S}$ of candidate services in Figure 2.5 with their property vectors. Service S_1 is not valid w.r.t. $\mathscr{C}^{\mathbb{A}}$ since it does not satisfy Conditions 1 and 2 of Definition 7. In fact, $\mathsf{pv}_{S_1}[\texttt{loc}] \notin Dom^+(\texttt{loc})$ and $\mathsf{pv}_{S_1}[\texttt{encr}] \notin \texttt{Dom}^+(\texttt{encr})$, with $Dom^+(\texttt{loc})$ ($Dom^+(\texttt{encr})$, respectively) constrained by R_1 (E_1, respectively). Condition 2 is violated by S_1 because of the value disjunction D_1: in fact, $\mathsf{pv}_{S_1}[\texttt{pTest}] = \text{int} \notin Dom(\texttt{pTest})|D_1 = \{\text{authA}, \text{authB}\}$, and also $\mathsf{pv}_{S_1}[\texttt{cert}] = \text{certC} \notin Dom(\texttt{cert})|D_1 = \{\text{certA}, \text{certB}\}$. On the contrary, service S_3 represents an example of a valid service w.r.t. $\mathscr{C}^{\mathbb{A}}$ as it satisfies all the conditions in Definition 7.

Figure 2.9 illustrates an algorithm for determining a set of valid services. It takes as inputs a set $\mathbb{S}$ of Cloud services characterized by a set $\mathbb{A}$ of attributes modeling properties of interest to the user, and a well-formed constraint policy $\mathscr{C}^{\mathbb{A}}$ over $\mathbb{A}$. The algorithm then returns the set $\mathbb{S}^v \subseteq \mathbb{S}$ of services that are valid (Definition 7) w.r.t. $\mathscr{C}^{\mathbb{A}}$. The algorithm operates in two steps, and in the first step (lines 1–6) it computes the constrained domain $Dom^+(A)$ for each attribute $A \in \mathbb{A}$ according to the rules in Figure 2.8. In the second step (lines 7–40) it evaluates each service $S \in \mathbb{S}$ to check its compliance with the conditions in Definition 7. To this aim, the algorithm starts by initializing variable $\mathbb{S}^v$ (which will include at the end the valid services) to the entire set $\mathbb{S}$ (line 7), and by iteratively removing from $\mathbb{S}^v$ all services that are not valid. The algorithm evaluates one service at a time (line 8), and checks the three conditions in Definition 7 in order, so that for each

service in $\mathbb{S}$, Condition 3 is checked only if Condition 2 is satisfied, which is in turn checked only if Condition 1 is also satisfied. This is done by checking the status of the Boolean variable *satisfied* (lines 15 and 25), which is set to TRUE only if the service under analysis satisfies the condition being checked. For each service S, Condition 1 is checked by checking whether there exists an attribute A assuming in pv_S a value not included in the constrained domain $Dom^+(A)$ computed in the first step: if this is the case, then variable *satisfied* is set to FALSE (line 12) not to check the other validity conditions on S, and S is then removed from $\mathbb{S}^v$ (line 13). Otherwise, S satisfies Condition 1 in Definition 7, and Condition 2 can be evaluated (lines 15–24). To this aim, the algorithm evaluates all value disjunctions in $\mathbb{D}$ in order. For each $D \in \mathbb{D}$, *satisfied* is first set to FALSE, and is set to TRUE only if there exists at least one attribute $A \in \mathbb{A}$ assuming a value as required by D (lines 19–20). If this is the case, the algorithm then terminates the evaluation of the attributes (since D is satisfied, line 21). If D is not satisfied by S (i.e., *satisfied* is still set to FALSE, line 22), then S is removed from $\mathbb{S}^v$ (line 23), and no further check is done on S. Otherwise, S satisfies Condition 2 in Definition 7, and Condition 3 can be finally evaluated (lines 25–39). To this aim, the algorithm evaluates all value implications in $\mathbb{I}$ in order, determining first whether an implication is not applicable to S (lines 30–32) and, if so, sets Boolean variable *implication* to FALSE. If the implication is applicable to S (line 33), then the algorithm checks if S also satisfies the tail of the implication: to this aim, it is sufficient that there exists an attribute in pv_S assuming a value acceptable by the implication (lines 34–36). If this is not the case, S is removed from $\mathbb{S}^v$ and no further check is done on S. Once the algorithm has iterated over all $S \in \mathbb{S}$, $\mathbb{S}^v$ (containing all valid services) is returned to the user (line 40).

With reference to our running example, executing the algorithm over the set $\mathbb{S} = \{S_1, \ldots, S_7\}$ of services in Figure 2.5, the set $\mathbb{A}$ of attributes in Figure 2.4, and the constraint policy $\mathscr{C}^{\mathbb{A}}$ in Figure 2.7, the set $\mathbb{S}^v$ of valid services will be $\mathbb{S}^v = \{S_3, S_4, S_5, S_6, S_7\}$. In fact, S_1 is removed from $\mathbb{S}^v$ when checking compliance with Condition 1, and S_2, while satisfying Condition 1, is removed from $\mathbb{S}^v$ when checking Condition 2. Note that, although S_1 does not satisfy Condition 2 as well, the algorithm will not check such compliance because S_1, failing the check on Condition 1, could not be a valid service anyway. The same reasoning applies also to service S_2, which does not satisfy Condition 3 but this is not checked by the algorithm because S_2 fails the check on Condition 2.

2.4.6 Enforcing User Preferences

Given a set $\mathbb{S}$ of Cloud services and a constraint policy $\mathscr{C}^{\mathbb{A}} = \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ defined over a set $\mathbb{A}$ of attributes modeling the characteristics of the different services in $\mathbb{S}$, the set $\mathbb{S}^v$ of services valid w.r.t. $\mathscr{C}^{\mathbb{A}}$ can include several services (see for example our running example where, out of seven candidate providers in $\mathbb{S}$, five are valid w.r.t. the constraint policy). While all valid services can be successfully considered for outsourcing (all valid services are in fact guaranteed to satisfy $\mathscr{C}^{\mathbb{A}}$), the user might still have preferences among them, for instance driven by the different guarantees they offer. As an example, a user might consider acceptable both the AES and 3DES ciphers (specifying for instance the value exclusion E_1 in Figure 2.7), while preferring AES to 3DES for the security guarantees it offers. Intuitively, if there exist two valid services such that one features AES (e.g., S_5 in Figure 2.5) and the other 3DES (e.g., S_4), then the one with AES would be preferred by the user. In this section, we illustrate how users can specify *preferences* over the characteristics of the services in $\mathbb{S}$. The definition of these preferences nicely complements our formulation of constraints and valid services, and can be used to assess the degree by means of which a set of valid services satisfy the needs of the user.

INPUT
$\mathbb{S} = \{S_1, \ldots, S_n\}$ /* set of Cloud services */
$\mathbb{A} = \{A_1, \ldots, A_m\}$ /* set of attributes characterizing $\mathbb{S}$ */
$\mathscr{C}^{\mathbb{A}} = \langle \mathbb{R}, \mathbb{E}, \mathbb{D}, \mathbb{I} \rangle$ /* constraint policy */

OUTPUT
$\mathbb{S}^v \subseteq \mathbb{S}$ /* set of valid services w.r.t. $\mathscr{C}^{\mathbb{A}}$ */

MAIN

/* **Step 1:** compute constrained domains */

1: **for each** $A \in \mathbb{A}$ **do**
2: **if** $\exists R \in \mathbb{R}$ s.t. $R = \langle A \text{ IN } \{v_1, \ldots, v_h\} \rangle$ **then**
3: $Dom^+(A) := \{v_1, \ldots, v_h\}$
4: **else if** $\exists E \in \mathbb{E}$ s.t. $E = \langle A \text{ NOT IN } \{v_1, \ldots, v_h\} \rangle$ **then**
5: $Dom^+(A) := Dom(A) \setminus \{*, v_1, \ldots, v_h\}$
6: **else** $Dom^+(A) := Dom(A)$

/* **Step 2:** find valid services */

7: $\mathbb{S}^v := \mathbb{S}$
8: **for each** $S \in \mathbb{S}$ **do**
9: *satisfied* := TRUE
10: **for each** $A \in \mathbb{A}$ **do** /* check Condition 1 in Def. 7 */
11: **if** $\mathsf{pv}_S[A] \notin Dom^+(A)$ **then**
12: *satisfied* := FALSE
13: $\mathbb{S}^v := \mathbb{S}^v \setminus \{S\}$
14: **break**
15: **if** *satisfied* = TRUE **then**
16: **for each** $D \in \mathbb{D}$ **do** /* check Condition 2 in Def. 7 */
17: let $\mathbb{A}^D$ be the set of attributes over which D is defined
18: *satisfied* := FALSE
19: **for each** $A \in \mathbb{A}^D$ **do**
20: **if** $\mathsf{pv}_S[A] \in Dom^+(A)|D$ **then** *satisfied* := TRUE
21: **if** *satisfied* = TRUE **then break**
22: **if** *satisfied* = FALSE **then**
23: $\mathbb{S}^v := \mathbb{S}^v \setminus \{S\}$
24: **break**
25: **if** *satisfied* = TRUE **then**
26: **for each** $I \in \mathbb{I}$ **do** /* check Condition 3 in Def. 7 */
27: let $\mathbb{A}^{I.h}$ be the set of attributes over which $I.h$ is defined
28: let $\mathbb{A}^{I.t}$ be the set of attributes over which $I.t$ is defined
29: *satisfied* := FALSE
30: *implication* := TRUE
31: **for each** $A \in \mathbb{A}^{I.h}$ **do**
32: **if** $\mathsf{pv}_S[A] \notin Dom(A)|I$ **then** *implication* := FALSE
33: **if** *implication* = TRUE **then**
34: **for each** $A \in \mathbb{A}^{I.t}$ **do**
35: **if** $\mathsf{pv}_S[A] \in Dom(A)|I$ **then** *satisfied* := TRUE
36: **if** *satisfied* = TRUE **then break**
37: **if** *satisfied* = FALSE $\wedge$ *implication* = TRUE **then**
38: $\mathbb{S}^v := \mathbb{S}^v \setminus \{S\}$
39: **break**
40: **return** $\mathbb{S}^v$

Figure 2.9: Function returning a set of valid Cloud services

loc	encr	rep	avail	pTest	cert	audit
EU	AES	4	99.5%	authA	certA	3M
US	3DES	3	99.2%	authB	certB	6M
		2	99%	int	certC	1Y
			97%, 98%	*	certD	*
					*	

Figure 2.10: Value preferences over the attributes in Figure 2.4

A natural way to express preferences in our scenario (which is also nicely in line with our modeling of constraints) operates on the different values that the attributes in $\mathbb{A}$ can assume, expressing whether the user prefers one value over another for a certain attribute A or, more generally, a certain set of values over another. To this aim, we introduce the concept of *value preference*, as follows.

Definition 8 (Value preference)**.** *Given a set $\mathbb{A}$ of attributes, an attribute $A \in \mathbb{A}$ with domain $Dom(A)$, and a set $Dom'(A) \subseteq Dom(A)$ of values, a* value preference *over A is a totally ordered partition $(Part(A), \succ)$ of $Dom'(A)$ such that:*

1. *$\forall P \in Part(A)$, all values in P are equally preferred;*
2. *$\forall P_i, P_j \in Part(A), P_i \neq P_j$ s.t. $P_i \succ P_j$, any value in P_i is preferred to any value in P_j.*

A value preference over an attribute A corresponds to a total ordering among a partition $\{P_1, \ldots, P_k\}$ defined over a set of values that A can assume, where the partitions are built in such a way to cluster together all values that are equally desirable to the user. Clearly, if all partitions include one element (i.e., there does not exist a set of elements equally desirable), then this corresponds to a total ordering among the values. Value preferences can be graphically represented through a simplified Hasse diagram with an element for each partition, and an upward edge linking partition P_j to P_i iff P_i *covers* P_j (i.e., $P_i \succ P_j$ and $\nexists P_h$ s.t. $P_i \succ P_h \succ P_j$, $P_h \neq P_i, P_h \neq P_j$). To illustrate, consider the value preference over attribute `loc` in Figure 2.10. In this example, the preference has been defined over the set of values $\{\text{EU}, \text{US}\} \subset Dom(\texttt{loc})$, partitioned in two partitions as $\{\{\text{EU}\}, \{\text{US}\}\}$, and states that value EU is preferred to value US to the user. We assume that, if the set of values over which a preference is to be defined includes the $*$ value to indicate no guarantee provided on that attribute (see Section 2.4.1), then the ordering will include a partition including $*$ only, and will be the bottom element of the ordering.

We refer to the set of value preferences specified by the user as her *preference policy*. More precisely, given a set $\mathbb{A}$ of attributes, a preference policy $\mathscr{P}^{\mathbb{A}}$ over $\mathbb{A}$ is a set of value preferences over a subset of $\mathbb{A}$. In the remainder of this document, we assume preference policies to be *well-formed*, as follows.

Definition 9 (Well-formed preference policy)**.** *Given a set $\mathbb{A}$ of attributes, a constraint policy $\mathscr{C}^{\mathbb{A}}$ over $\mathbb{A}$, and a preference policy $\mathscr{P}^{\mathbb{A}}$ over $\mathbb{A}$, $\mathscr{P}^{\mathbb{A}}$ is said to be* well-formed *w.r.t. $\mathscr{C}^{\mathbb{A}}$ iff the following three conditions hold:*

1. *$\forall A \in \mathbb{A}$, $\mathscr{P}^{\mathbb{A}}$ includes a value preference over A;*
2. *$\forall A \in \mathbb{A}, \forall P \in Part(A), \forall v \in P : v \in Dom^{+}(A)$;*

3. $\forall A \in \mathbb{A}, \forall v \in Dom^{+}(A) : \exists P \in Part(A)$ *s.t.* $v \in P$.

Definition 9 states that: a well-formed preference policy $\mathscr{P}^{\mathbb{A}}$ over a set $\mathbb{A}$ of attributes includes one value preference for each attribute in $\mathbb{A}$ (Condition 1); the partitions in which each attribute of $\mathbb{A}$ is divided by $\mathscr{P}^{\mathbb{A}}$ only include values considered acceptable by the constraint policy $\mathscr{C}^{\mathbb{A}}$ (Condition 2), as it would in fact be useless for the user to specify preferences over values that she does consider acceptable; and preferences are complete (Condition 3), as otherwise a valid service with a value with no preference could not be assessed. It is immediate to see that the preference policy graphically illustrated in Figure 2.10 is well-formed w.r.t. the constraint policy in Figure 2.7.

Equipped with the notion of preference policy, we now illustrate how a policy $\mathscr{P}^{\mathbb{A}}$ can be enforced on a set $\mathbb{S}^{v}$ of valid services, to the aim of determining the degree by means of which each service in $\mathbb{S}^{v}$ satisfies $\mathscr{P}^{\mathbb{A}}$. To this end, we propose two strategies, starting with the the most intuitive approach SK-dominance, and then proposing a refinement D-dominance. For each approach, we will refer our discussion to the problem of comparing two services to the aim of determining which one better satisfies the preferences defined by the user.

SK-dominance. The first strategy we propose to compare two Cloud services is based on the well-known concept of dominance among points in a multidimensional space. Informally, given two points p and q, *p dominates q* if p is better than or equal to q in all dimensions, and better than q in at least one dimension; p is a *skyline point* iff there does not exist any other point that can dominate p (hence, our naming SK-dominance, from *Skyline dominance*). In our framing of the problem, points represent valid services (i.e., we have $|\mathbb{S}^{v}|$ points) and a dimension represents a value preference over an attribute (i.e., we have $|\mathbb{A}|$ dimensions). Formally, SK-dominance among a pair of services is defined as follows.

Definition 10 (SK-dominance)**.** *Given a set $\mathbb{A}$ of attributes, a constraint policy $\mathscr{C}^{\mathbb{A}}$ over $\mathbb{A}$, a preference policy $\mathscr{P}^{\mathbb{A}}$ over $\mathbb{A}$ well-formed w.r.t. $\mathscr{C}^{\mathbb{A}}$, and two Cloud services S_x, S_y valid w.r.t. $\mathscr{C}^{\mathbb{A}}$ with pv_{S_x} and pv_{S_y} their property vectors, S_x SK-dominates S_y (denoted $S_x >_{\mathsf{SK}} S_y$) iff the following two conditions hold:*

1. $\forall A \in \mathbb{A} : P_x \succeq P_y$,
2. $\exists A \in \mathbb{A} : P_x \succ P_y$

where P_z is the partition of $Dom(A)$ including $\mathsf{pv}_{S_z}[A]$, $\succ$ the value preference in $\mathscr{P}^{\mathbb{A}}$ over A, and $P_h \succeq P_k$ iff $P_h \succ P_k$ or $P_h = P_k$.

According to Definition 10, given two valid services S_x and S_y such that $S_x >_{\mathsf{SK}} S_y$, then S_x enjoys a value v_i for at least one attribute A in $\mathbb{A}$ that is more preferred than the value $v_j \neq v_i$ assumed by S_y for A, while not assuming for all other attributes values that are less preferred than those assumed by S_y. Clearly, if these conditions are satisfied, then S_x satisfies the preference policy better than S_y, and can therefore be preferred to S_y. To illustrate, consider services S_3 and S_5 in Figure 2.5 and the constraint and preference policies in Figures 2.7 and 2.10 (note that both S_3 and S_5 are valid w.r.t. the constraint policy in Figure 2.7). According to the conditions in Definition 10, we have that $S_5 >_{\mathsf{SK}} S_3$. In fact, Condition 2 in Definition 10 is satisfied as S_5 assumes better values than S_3 for attributes `rep` and `pTest` (i.e., $\mathsf{pv}_{S_3}[\texttt{rep}] = 2$; $\mathsf{pv}_{S_5}[\texttt{rep}] = 3$; $\mathsf{pv}_{S_3}[\texttt{pTest}] = \texttt{authB}$; $\mathsf{pv}_{S_5}[\texttt{pTest}] = \texttt{authA}$). Condition 1 is also satisfied since S_3 does not assume, for all other attributes, a value preferred to those assumed by S_5.

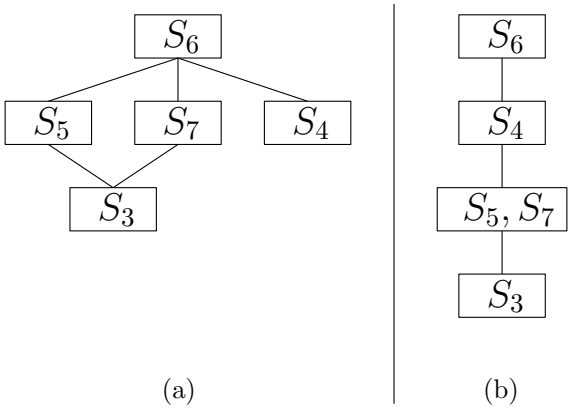

Figure 2.11: Preference enforcement according to SK-dominance (a) and D-dominance (b)

Given the set $\mathbb{S}^v \subseteq \mathbb{S}$ of valid Cloud services, the SK-dominance in Definition 10 naturally induces a hierarchy $(\mathbb{S}^v, >_{\mathsf{SK}})$ over $\mathbb{S}^v$, with an element for each service in $\mathbb{S}^v$ and an upward edge between two services $S_y, S_x \in \mathbb{S}^v$ iff S_x covers S_y according to the SK-dominance operator $>_{\mathsf{SK}}$. To illustrate, consider the set $\mathbb{S}$ of services in Figure 2.5, the constraint and preference policies in Figures 2.7 and 2.10, and the set $\mathbb{S}^v = \{S_3, S_4, S_5, S_6, S_7\}$ of valid services. According to the SK-dominance in Definition 10, we have that $S_6 >_{\mathsf{SK}} S_3$, $S_6 >_{\mathsf{SK}} S_4$, $S_6 >_{\mathsf{SK}} S_5$, $S_6 >_{\mathsf{SK}} S_7$, $S_5 >_{\mathsf{SK}} S_3$, $S_7 >_{\mathsf{SK}} S_3$. Figure 2.11(a) graphically illustrates the hierarchy $(\mathbb{S}^v, >_{\mathsf{SK}})$ induced by the $>_{\mathsf{SK}}$ operator.

D-dominance. The hierarchy induced by the SK-dominance operator $>_{\mathsf{SK}}$ (Definition 10) naturally corresponds to a partial ordering among the set $\mathbb{S}^v$ of valid services. While certainly representing a possible way of enforcing a preference policy, being a partial ordering it might include pairs of incomparable services, for which nothing can be said w.r.t. preference satisfaction. With reference to the ranking illustrated in Figure 2.11(a), for example, nothing can be said on pairs $\langle S_5, S_7\rangle$, $\langle S_5, S_4\rangle$, $\langle S_7, S_4\rangle$, and $\langle S_4, S_3\rangle$ of services. This is because the SK-dominance is not guaranteed to be definable over every service pair (i.e., given two services, it is not guaranteed that one dominates the other). To provide the user with more informative results, we propose a second approach that assesses *how much* the value preferences are satisfied by the services in $\mathbb{S}^v$.

To this aim, we associate each service $S \in \mathbb{S}^v$ with a *score vector* $\mathsf{sv}_S[A_1, \ldots, A_m]$ where each value $\mathsf{sv}_S[A_i]$ includes a *score* of the value $\mathsf{pv}_S[A_i]$ based on its position in the Hasse diagram of $Dom(A_i)$, with pv_S the property vector of service S. Given value $\mathsf{pv}_S[A]$, its score $\mathsf{sv}_S[A]$ is defined as $1 - \frac{i}{k}$, with k the cardinality of $Part(A)$ and i the number of sets in $\{P_j \in Part(A) : P_j \succ P_i, \mathsf{pv}_S[A] \in P_i\}$. Figure 2.12 illustrates in the first five rows a tabular representation of the score vectors of the services in Figure 2.5 safe w.r.t. the constraint policy in Figure 2.7, and induced by the value preferences in Figure 2.10. For instance, $\mathsf{sv}_{S_3} = [1/2, 1, 1/3, 3/4, 3/4, 3/5, 1/4]$, representing the fact that, for instance, the value assumed by S_3 for attribute `pTest` (i.e., $\mathsf{pv}_{S_3}[\mathtt{pTest}] =$ authB) has a score in the Hasse diagram of the value preferences over `pTest` (Figure 2.10) of $\mathsf{sv}_{S_3}[\mathtt{pTest}] = 3/4$.

Given a service S with its score vector sv_S, an intuitive way of assessing the degree by means of which S satisfies the value preferences over the attributes in $\mathbb{A}$ is to consider the score vectors of the services as their coordinates in a m-dimensional space (with $m = |\mathbb{A}|$), and to evaluate the Euclidean distance between sv_S and a *baseline* score vector $\mathsf{sv}_{S^\top}$, defined to represent the (possibly not existing) baseline Cloud service $S^\top$ that best satisfies them. Such possibly fictitious

	loc	encr	rep	avail	pTest	cert	audit
S_3	1/2	1	1/3	3/4	3/4	3/5	1/4
S_4	1	1/2	2/3	1	3/4	4/5	1/4
S_5	1/2	1	2/3	3/4	1	3/5	1/4
S_6	1	1	2/3	1	1	1	1/4
S_7	1/2	1	2/3	1	3/4	3/5	1/4
$S^\top$	1	1	1	1	1	1	1

Figure 2.12: Tabular representation of the score vectors of a set of safe services (S_3,...,S_7) and of the baseline service $S^\top$

service $S^\top$ will intuitively assume, for each attribute in $\mathbb{A}$, the best value possible according to the value preferences set by the user. For instance, w.r.t. the value preferences in Figure 2.10, $\mathsf{pv}_{S^\top} = [\text{EU}, \text{AES}, 4, 99.5\%, \text{authA}, \text{certA}, 3\text{M}]$, i.e., the top elements in the value preferences for all attributes. Note that, when for a certain attribute the top element is included in a non-singleton partition of $Dom^+(A)$, any value would do as they all are equivalent (Condition 1 in Definition 8). As a consequence, $\forall A \in \mathbb{A}$, the baseline score vector $\mathsf{sv}_{S^\top}$ of $S^\top$ will have value 1 (i.e., $\forall A \in \mathbb{A}, \mathsf{sv}_{S^\top}[A] = 1$). The last row in Figure 2.10 illustrates the score vector $\mathsf{sv}_{S^\top}$ of the baseline service $S^\top$. Intuitively, the Euclidean distance between the score vectors of a valid service S_x and that of the baseline service $S^\top$ quantifies how much S_x 'differs' from $S^\top$ and, given two services S_x and S_y, the one with score vector closer to $S^\top$ will better satisfy the preferences of the user, as follows.

Definition 11 (D-dominance)**.** *Given a set $\mathbb{A}$ of attributes, a constraint policy $\mathscr{C}^\mathbb{A}$ over $\mathbb{A}$, a preference policy $\mathscr{P}^\mathbb{A}$ over $\mathbb{A}$ well-formed w.r.t. $\mathscr{C}^\mathbb{A}$, and two Cloud services S_x, S_y valid w.r.t. $\mathscr{C}^\mathbb{A}$ with sv_{S_x} and sv_{S_y} their score vectors, S_x D-dominates S_y (denoted $S_x >_\mathsf{D} S_y$) iff the following condition holds:*

$$dist(\mathsf{sv}_{S_x}, \mathsf{sv}_{S^\top}) < dist(\mathsf{sv}_{S_y}, \mathsf{sv}_{S^\top})$$

with $dist(\mathsf{sv}_h, \mathsf{sv}_k)$ the Euclidean distance between sv_h and sv_k.

Let us recall that, in our framing of the problem, the Euclidean distance $dist(\mathsf{sv}_x, \mathsf{sv}_y)$ between a pair of score vectors is defined as

$$dist(\mathsf{sv}_x, \mathsf{sv}_y) = \sqrt{\sum_{i=1}^{|\mathbb{A}|} (\mathsf{pv}_{S_x}[A_i] - \mathsf{pv}_{S_y}[A_i])^2}$$

Given two valid services S_x and S_y such that $S_x >_\mathsf{D} S_y$, then S_x satisfies the preference policy $\mathscr{P}^\mathbb{A}$ defined by the user more than what is done by S_y, since S_x is closer to the best possible service $S^\top$ than S_y in an m-dimensional space (with m the number of attributes in $\mathbb{A}$). To illustrate, consider score vectors sv_{S_4} and sv_{S_6} in Figure 2.12. We have that $dist(\mathsf{sv}_{S_4}, \mathsf{sv}_{S^\top}) = 1.0129 > dist(\mathsf{sv}_{S_6}, \mathsf{sv}_{S^\top}) = 0.8207$, and therefore S_6 better satisfies the preference policy than S_4. Figure 2.11(b) graphically illustrates the enforcement of the preference policy in Figure 2.10 according to the $>_\mathsf{D}$ dominance operator over the set $\{S_3, \ldots, S_7\}$ of valid services.

Note that, unlike the skyline dominance $>_\mathsf{SK}$ in Definition 10, whenever the distance dominance $>_\mathsf{D}$ in Definition 11 is not defined for a pair of services S_x and S_y (i.e., $S_x \not>_\mathsf{D} S_y$ and $S_y \not>_\mathsf{D} S_x$), we have the guarantee that the distances between the score vectors of S_x and S_y and that of the baseline service $S^\top$ are *equal*, and we can therefore conclude that S_x and S_y can be regarded as being *equivalent* w.r.t. the satisfaction of $\mathscr{P}^\mathbb{A}$. In fact, the distances can be equal only if: *i)* S_x

and S_y, while being different services, have the same property set; or *ii)* S_x and S_y have different properties but the ameliorations provided to some attributes are compensated by a worsening of other attributes. For instance, consider Figure 2.11(b). Since $dist(\mathsf{sv}_{S_5}, \mathsf{sv}_{S^\top}) = dist(\mathsf{sv}_{S_7}, \mathsf{sv}_{S^\top}) = 1.0705$, we can conclude that S_5 and S_7 satisfy $\mathscr{P}^{\mathbb{A}}$ equally. In fact, it is easy to see that, despite having same values for attributes `loc`, `encr`, `rep`, `cert`, and `audit`, the better value assumed by S_5 for `pTest` (i.e., authA as opposed to authB for S_7) is 'compensated' the worse value assumed for `avail` (i.e., 99.2% as opposed to 99.5% for S_7). In this regard, we can see the enforcement of a preference policy through $>_{\mathsf{D}}$ in Definition 11 as an ordering among the services in $\mathbb{S}^v$.

The two dominance we have illustrated so far are naturally related, and it is immediate to see that SK-dominance $>_{\mathsf{SK}}$ implies the D-dominance $>_{\mathsf{D}}$: for this reason, given two Cloud valid services S_x and S_y, we have that $S_x >_{\mathsf{SK}} S_y \Longrightarrow S_x >_{\mathsf{D}} S_y$.

We close this section with a note that, besides value preferences, users might be wishing to express other kinds of soft requirements, for instance based on the properties themselves (for instance, to specify that they consider a certain property/attribute in our modeling more important than another one). We are currently investigating into extending our approach to the consideration of additional kinds of preferences, as well as to enhancements in the way constraints can be specified, to further augment their expressive power.

2.5 Summary

This chapter illustrated the models and techniques developed in Task 4.2 for leveraging multiple providers for security. Section 2.2 reported on the integration of the distributed shuffle index with a real industrial scenario, showing its promising performance. Section 2.3 introduced an analysis on the use of distributed Cloud storage, observing how an AONT transformation could effectively provide security guarantees. Section 2.4 illustrated our approach for allowing users to specify requirements and preferences in the selection of the Cloud services to be adopted in a multi-provider scenario.

3. Data Protection as a Service for Federated Cloud Storage

The main focus of the work done under Task 4.3 of the ESCUDO-CLOUD was to design and develop a set of security tools, techniques and components that can be used to compose and implement a Data Protection as a Service (DPaaS) solution. The goal of this solution was to allow BT customers to protect and control their confidential and sensitive information with a user-friendly data protection service, that keeps their data private and helps meet regulatory compliance requirements. Customers should be able to store their data on multiple Cloud platforms, and should be able to manage the security related aspects of their stored data via the federated protection service. Only the customers (or a trusted third party designated by the customers) should have access to and control of the cryptographic keys, giving the freedom to decrypt data on-demand and in real-time.

The main challenge addressed in this task was the security and management of data that is hosted on third party infrastructures, for example in the form of block/file-system storage, data backups, or databases. This problem is further complicated in the Cloud computing environment as data can be replicated and moved automatically to cater for the scalability and reliability needs of the Cloud providers' customers, thus increasing the risk of a security compromise. In addition to the data security concerns, most customers also have to abide by their company's data protection policies and governmental regulatory compliance (e.g., FIPS, HIPAA, HITECH, Sarbanes-Oxley, GLB, PCI DSS and GDPR etc.).

Furthermore, a customer can make use of many other Cloud service providers offering storage services, in addition to BT Cloud Compute, with each Cloud provider offering its own API, specialized services, and security functionalities to satisfy various user requirements. Therefore, the constituent components of DPaaS should be able to construct a Cloud security service that provides data protection service to a customer for block and object storage and Big Data services, and works seamlessly across multiple Cloud platforms.

The rest of this chapter is structured as follows: Section 3.1 presents a compact overview of the contributions within Task 4.3 and the work on those innovations. Section 3.2 presents a brief summary of the work carried out within Task 4.3 until M24 and presented in the first version of this deliverable (D4.2). Section 3.3 provides an overview of the reference tools and techniques identified and constructed for Use Case 3. A description of each component is provided, as well as their salient functions and features, their purpose and motivation in the architecture of the solution, and their role within the scope of Use Case 3. Section 3.4 describes the design of the current high level architecture, the granularity of the main components, and the approach towards final solution design. Section 3.5 provides the implementation details about the prototype developed to show the feasibility of the proposed solution. Section 3.6 briefly describes the integration between the BT DPaaS and the EncSwift solution (developed by UNIBG as part of the work done in WP 2 of ESCUDO-CLOUD). Section 3.7 draws a few concluding remarks.

3.1 ESCUDO-CLOUD Innovations

The key innovative features of the Data Protection as a Service model as proposed in the context of ESCUDO-CLOUD are:

- An independent service offered via the Cloud service store that enables customers to protect data stored on multiple Cloud platforms.
- The data is protected by encrypting it, and ensuring that Cloud service providers have no access to the encryption keys or protection policies.
- The encrypted data can be stored on multiple Cloud storage services like virtual volumes, object stores and Big Data clusters etc.
- Access control and key management are offered as independent but tightly coupled services that manage the protection of the data via an integrated policy framework.

3.2 D4.2 Summary - First Report on Multi-Cloud and Federated Cloud

D4.2 covered the M24 status of the design and development of the DPaaS solution. It listed and briefly described the techniques identified and selected for the DPaaS solution. The description was in form of the functions and features of the tools and techniques, their purpose and place in the overall vision of the DPaaS solution and their specific application to the ESCUDO-CLOUD Use Case 3.

D4.2 also showcased the design of the high-level DPaaS architecture. It describes the scope and granularity of some of the main components of the solution, especially those related data protection for Cloud *block* and *object* storage services. One of the main focuses of the design of the DPaaS solution is the customer ownership and control of the outsourced data. This was achieved by the integration of key management and access control features, in which only the customers can exercise the control over the encryption keys. Only they have the ability to regulate access to their data through policy based enforcement of access control attributes.

As the customers are provided with the choice of selecting storage services from multiple CSPs, by retaining control of key management the customers have more flexibility in meeting regulatory and privacy requirements and ensuring data confidentiality and secure access. Thus, by utilizing the BT Service Store and its features, the Data Protection as a Service was able to support different security solutions in a Multi-Cloud environment. The BT Service Store provides full automation of data encryption services to users as a complete life-cycle, from data encryption to data decryption.

3.3 Tools and Techniques

In this section, we provide an overview of the main security tools and techniques that we have identified and selected in order to develop and implement the DPaaS. The core software components of the ESCUDO-CLOUD data protection service are the agents, tools and utilities providing the core data encryption/decryption mechanisms, the access control service and the key management service. The customers are able to use these services via the BT Service Store that integrates the

protection system and its components with multiple independent Cloud service providers. Therefore, the BT Service Store's orchestration capability is the central component of the data protection service.

3.3.1 Service Orchestrator

Purpose

The main purpose of the BT Service Store's Service Orchestrator is to provision the data protection service to the customers and manage its life-cycle. Each customer gets compartmentalized access to the data protection service, as discussed above, and is able to define the access control and key release policies via the Service Store or the key management service interface and associate keys with those policies. The Service Store also has the ability to install and configure the encryption agent software on virtual machines and gateways on different supported Cloud platforms.

Functions and features

The main functions and features offered by the BT Service Orchestrator are as follows:

- Abstraction of Cloud and infrastructure level details.
- A scalable repository to store all service related information, including all components, property settings.
- Management of relationships between multiple service components and all of their settings and configuration data.
- Completely automated Multi-Cloud deployment.
- A user friendly centralized management portal to simplify the process provisioning service components into multiple Cloud targets, and managing the service life-cycle seamlessly.

Application in Use Case 3

One of the core requirements of Use Case 3 of the ESCUDO-CLOUD is the ability to support the data security capabilities across multiple Cloud environments. This is where the Service Orchestrator [DDD+16] comes into use as it is designed in a manner so that the Cloud infrastructure and operating system level dependencies are decoupled. The main way this decoupling is achieved is by having different Cloud Management (CM) profiles that are tied to individual Cloud service provider APIs and operating system level operations being managed by a separate Configuration Management System (Puppet [Loo11] in this case). This enables the Service Orchestrator to achieve the goal of provisioning the data protection service into a wide variety of Cloud platforms, both public and private, as well as different kinds of operating systems hosted on them. Currently the BT Service Store supports deployments and life-cycle management of data protection service on Amazon Web Services EC2, Rackspace Cloud and Microsoft Azure in the public Cloud category, and Citrix CloudStack, CA Applogic and VMware vCloud in the private Cloud category. Currently supported operating systems are all current versions of Microsoft Windows and Linux distributions of Red Hat Enterprise Linux and Ubuntu.

3.3.2 Key Management Service

Purpose

The main purpose of the BT Key Management Service is to allow the customers to generate, store and manage their encryption keys and certificates securely. It is a centralized, high-availability and standards-based key management solution. It offers an intuitive web-based management console for enterprise-wide administration and audit of encryption keys. It is provisioned and managed via the BT Cloud Service Store's Service Orchestrator, which enables each customer to create isolated and compartmentalized key management domains, so that they can store and manage their keys used in multiple Cloud platforms in a secure multi-tenant environment. Lastly, it is tightly integrated with the BT Access Control Service that governs the rules and policies of key release to the authorized users and processes.

Functions and features

The main functions and features offered by the BT Key Management Service are following:

- Simplified multi-tenant and centralized key management.
- Generation and management of symmetric and asymmetric encryption keys (3DES, AES, RSA).
- Support multiple X.509 digital certificate formats like PKCS#7, PKCS#8, DER, PEM, and PKCS#12.
- Supports of multiple API standards like PKCS#11, Microsoft Extensible Key Management (EKM) and OASIS KMIP, which helps with automated and authorized key release.
- A user friendly centralized management portal to simplify the management of the keys' and certificates' life-cycle seamlessly.

Application in Use Case 3

One of the core requirements of Use Case 3, and in the broader context of ESCUDO-CLOUD, is to empower the end-users with maximum control and ownership of their data in the Cloud ecosystem. This is realized in Use Case 3 by the customer based management of the cryptographic keys, so that only they are able to access the key store and only they are able to authorize the release of keys to trusted encryption agents. The BT Service Store has no view or control of the customers' keys and other security credentials, even if the key management service is deployed on the BT Cloud platform.

3.3.3 Access Control Service

Purpose

The main purpose of the BT Access Control Service is to allow its customers the ability to regulate the access to their encrypted data stored on multiple Cloud platforms through a policy based enforcement of rich access control attributes. Thus its core value is to give the customers the assurance that their data remains protected from honest-but-curious Cloud service providers. The Access Control Service is a centralized and high availability access control solution that offers an

intuitive web-based management console for enterprise-wide policy declaration and management. It is tightly integrated with the BT Key Management Service (KMS), so that the customers can define access control and key release policies via the BT service store or the KMS interface and bind keys with those policies. This consolidation of key management and access control enables consistent policy implementation between customers and multiple Cloud service providers.

Functions and features

The main functions and features offered by the BT Access Control Service are as follows:

- Availability of comprehensive access controls, especially for block storage and file-system based Cloud storage services.
- Ability to enforce the principle of least privilege by using hierarchical access management policies.
- Granular privileged user access management policies, that can be applied by user, process, file type, time of day, and other parameters.
- A user friendly centralized management portal to simplify the management of the life-cycle of access control policies.

Application in Use Case 3

One of the core requirements of Use Case 3, as well as the main objective of ESCUDO-CLOUD, is to enable the end-users to exercise maximum control and ownership of their data in the Cloud eco-system. This is realized in Use Case 3 by the Access Control Service, by enabling the creation of a strong separation of duties between privileged Cloud service administrators and data owners outsourcing their data on these Cloud platforms. The data encryption agents encrypt files and objects etc. on the Cloud storage services, but leave their metadata in clear. In this way, IT administrators, including hypervisor, Cloud, storage, and server administrators, can perform their system administration tasks, without being able to gain privileged access to the sensitive data residing on the systems they manage. When the data owners and authorized customers want to access their data, the access control service lets them achieve this seamlessly by releasing the correct keys to the trusted encryption agents or gateways.

3.3.4 Data Encryption Agents

Purpose

The main purpose of the BT Data Encryption Agents is to offer capabilities for data-at-rest encryption of sensitive data stored on different types of Cloud storage services, as well as enforce privileged user access control on that data. It is a portable solution that can encrypt and protect data residing in physical, virtualized, object, and big data storage environments. It implements a transparent encryption approach at the filesystem level or volume level, so that there is no modification needed in the applications, infrastructures, or business practices that require access to the data.

Functions and features

The main functions and features offered by the BT Data Encryption Agents are following:

- Broad platform portability, supporting Windows, Linux, and UNIX operating systems.
- Transparency to all applications.
- High performance encryption, as well as support for hardware-based encryption acceleration products, such as Intel/AMD AES-NI and IBM P8 cryptographic coprocessor.
- Logging of all permitted, denied, and restricted access attempts from users, applications, and processes.

Application in Use Case 3

One of the core requirements of Use Case 3 is that the customers should be able to cache their keys on trusted virtual machines or gateways in order to outsource or improve performance of the encryption and decryption process. The BT Data Encryption Agents are the components that are considered trustworthy in the scope of Use Case 3 as they are provisioned and deployed by the BT Service Orchestrator in a secure process on Cloud hosted virtual machines. The agents themselves run and operate in a sand-boxed environment within the virtual machine that cannot be accessed or modified by the Cloud or even the virtual machine's administrators. The keys are obtained by the agents after proper authentication and authorization procedures and are only cached securely for the duration of the encryption or decryption process and are not present on the virtual machines or the gateways while the data is at rest.

3.4 Architecture of the Data Protection-as-a-Service

The DPaaS solution is designed around the core functional requirements of a multi-tenant customer scenario that aims to protect different types of data assets on multiple Cloud storage services. The main component that addresses the administration and management of multiple tenants/customers, multiple Cloud platforms and multiple Cloud storage services is the Service Orchestrator component described in the previous section. This is the central managerial component of the BT Service Store that is used to provision all the DPaaS capabilities to the BT customers, as shown in Figure 3.1. Each customer gets a compartmentalized access to the service store and the data protection service, as discussed in the previous section, and is able to define the access control and key release policies via the service store or the Key Management Service (KMS) interface. The BT Service Store also has the ability to install and configure the data encryption agents, plug-ins and gateways on virtual machines on different supported Cloud platforms.

The main benefit of this design is that a centrally managed data protection service is used to federate the disparity of different Cloud storage services on multiple Cloud platforms. In ESCUDO-CLOUD Use Case 3, as depicted in Figure 3.1, we focus on three different storage media, that are commonly requested by BT customers, to store and process data on current Cloud platforms. We manage this diversity of storage technologies as three different branches or three sub-use cases. The core technical use case is to offer data protection as a service in a multi-Cloud environment to the BT customers, whereas the service store provides the interface for the customers of the DPaaS to access and manage these storage services. As a result of this structure,

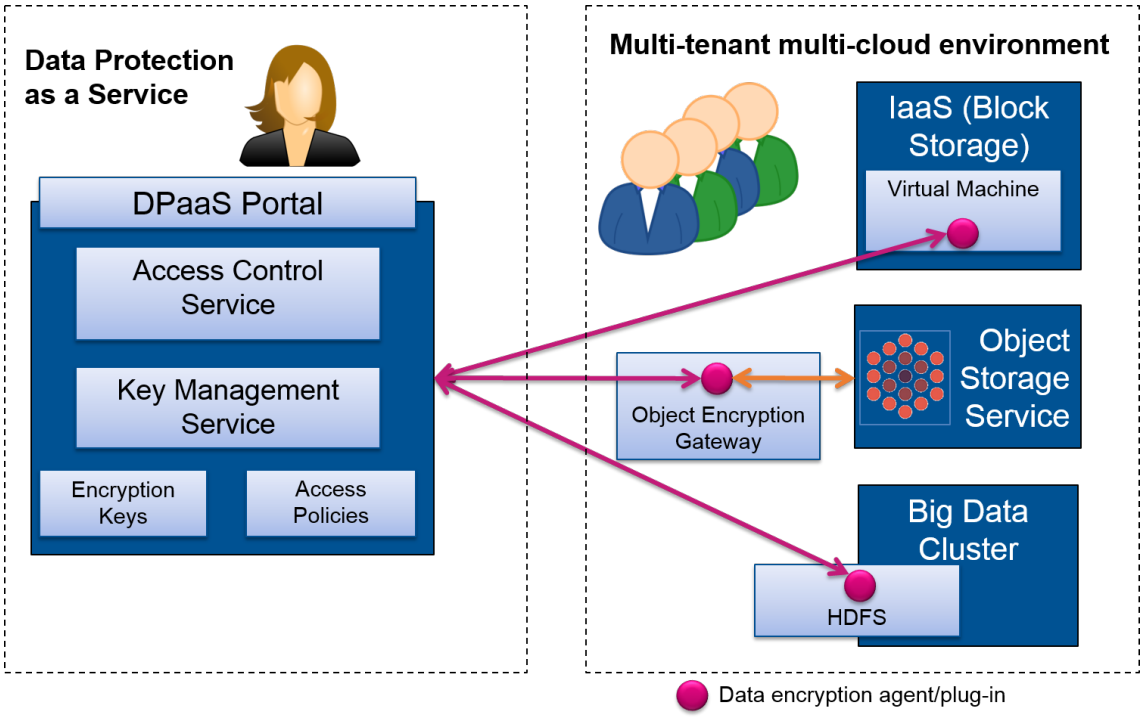

Figure 3.1: High level view of the DPaaS solution design

the sub-use cases will inherit a common encryption, key management and access control system but will have different abstractions and interfaces to cater for the provisioning, management and operation of underlying storage mediums. The architectural and implementation details of these three sub-use cases has been described in the next section.

Key Management Service

Controlling and maintaining the encryption keys is arguably the most important part of an organization's or customer's data protection strategy. Although encryption methods and algorithms are usually standardized and well understood, key management is often unique to each organization or customer. Nevertheless, most organizations have gradually adopted disparate systems with different ways of managing encryption keys over many years. Organizations are motivated to implement key management processes by any number of reasons, such as, a security breach event; the burden of managing disparate encryption systems; the impact of a lost encryption key that renders data inaccessible; or an industry compliance regulation. However, even after adopting a centralized key management solution, a key challenge for organizations and customers within this domain is ensuring the flexibility, scalability and interoperability of keys and key management systems, especially when interacting with other internal and external software components and services.

The Key Management Service in BT's DPaaS solution provides a robust and standards-based platform for managing encryption keys from disparate sources across an organization. It ensures that keys are stored securely and are always available to only authorized encryption/decryption services. It also provides the ability to audit and report on all activities relating to keys including key generation, rotation, destruction, key import, and key export. It also offers a 'Key Vault' to its users, which provides high availability storage and backup of symmetric and asymmetric encryption keys and tracks keys' expiration dates. Each instance of the Key Management Service

contains a public/private key pair ("master" key pair) used to protect the data encryption keys and other sensitive security objects stored within the Key Vault. The DPaaS administrators can change this key without any impact on the environment. In fact, as per PCI DSS requirements, the recommended best practice is to change the "master" key pair at least annually.

In the context of ESCUDO-CLOUD, the DPaaS allows for the application of a unique data encryption key for each virtual machine that accesses sensitive data, apply a single data encryption key across all virtual machines, or anything in between. The data owner, i.e., the customer is in full control of creating and assigning data encryption keys to balance the desire to minimize the impact of a compromised key with the need to minimize the administrative overhead of the annual key rotations. Some use cases that try to showcase this flexibility and control are:

- A BT customer offering a hosted document management system chooses to create and assign a data encryption key to databases and a separate data encryption key for unstructured data outside of the databases. The motivating factors for this approach might be the network location of the database servers versus the file and web servers as well as the volume of data in the databases versus the other systems.

- A BT customer that provides outsourced data processing chooses to create and assign a data encryption key for each of their *N* tenants.

- A BT customer with multiple, small business units chooses to create and assign data encryption keys for each of the business units. If every business unit has a limited number of systems, then this approach provides the balance between data segregation and administrative overhead. To allow for future expansion, the customer can assign unique keys per application within each business unit.

Access Control Service

The DPaaS Access Control Service provides robust role separation to allow organizations to securely leverage the Cloud storage infrastructure. Access Control administration can be broken down into responsibilities, so that one user might administer the creation of data encryption keys while a different user would administer the hosts and policies applied to that key. This separation can be taken further with access management domains that can combine this role-based administration with the ability to compartmentalize the management for policies, data encryption keys and agent configurations for a particular customer or tenant. For example, many enterprises need to compartmentalize data security management of corporate assets between business units and departments into different domains with different groups of security, key, and host administrators, while a smaller business may only have one security administrator for the entire deployment. Data security management can now be both centrally and departmentally administered.

DPaaS customers and tenants can apply access control policies to ensure system administrators and root users can manage the virtual machines and perform backups etc. without being able to view sensitive data. This feature is provided by the Data Encryption Agents, which allow management of data transparently by encrypting file content data but not filesystem metadata. By leaving the filesystem metadata in the clear, data management applications and system administrators can perform necessary functions without the need to expose file content during management operations. When encrypting with in-line encryption methods, companies are forced to open full data access to system administrators and root users of the operating systems, in order to complete their routine IT administration tasks, simply trusting these administrators not to compromise the

confidentiality of the data. By leveraging this approach, we can offer customers Cloud storage services according to a 'honest-but-curious' trust model.

The Access Control Service follows a least-privilege model, which means that any attempt at data access that is not specifically authorized according to well-defined, pre-set parameters are blocked. This added layer of optional access controls plus encryption enables customers to protect against a broad set of data security threats. More specifically, granular privileged user access management policies can be applied with respect to users, processes, file types, file attributes, time-of-day, and other parameters. These enforcement options are very granular and they can be used to control not only the permission to access plaintext data, but also what filesystem operations are available to the authorized users.

Data Encryption Agents

Data Encryption Agents provide the capability of data-at-rest encryption for the block, object and Big Data storage services. They are provisioned, installed and configured by the BT service store's Service Orchestrator on the virtual machines hosting the block storage, Big Data clusters (NameNodes and DataNodes) and on BT's encryption gateways for the object storage encryption service. These agents are also the enforcement points of the access control policies defined by the data owners using the Access Control Service. Using these agents, customer organizations can implement and enforce data protection with minimal disruption, as their operation is transparent to the customer's existing applications, databases or infrastructure. This transparency is ensured by using different types of keys, for performing different levels of data encryption operations and for providing confidentiality and integrity of communication between the agents and the Key Management and Access Control services. The types and purpose of these keys is given below:

Agent Key Pair: Each Data Encryption Agent has a public/private key pair (AKP), which is used as one of the parameters to protect the data encryption keys in transit and, if configured for offline key cache, at rest on the local virtual machine. The DPaaS administrators can change these keys by re-registering the agent. This operation does not require a protection service outage, it only requires a restart of the agent process.

Key Encryption Key: A random AES 256-bit Key Encryption Key (KEK) is used to protect the Data Encryption Keys (DEK) in transit from the Key Management Service to the Data Encryption Agents. This key is also used to encrypt the DEK at rest on the local virtual machine if using offline key cache (cached on Host feature). This AES encryption key is randomly generated by the Key Management Service each time the agent restarts and successfully initializes with the Key Management Service. Therefore, at a minimum the key is changed every time the agent protected system reboots.

The agent uses its AKP to encrypt/decrypt the KEK that is used to encrypt the DEK. The encryption is done using the public key at the KMS, the encrypted key bundle (including the KEK and DEK) are sent to the agent, and the decryption is carried out using the private key. Now all relevant keys are with the agent.

Data Encryption Key: The Data Encryption Keys are never exposed to users, administrators, or applications, and are always encrypted at rest and in transit. Further, when the DEK is decrypted for use by the agent, it is kept in non-swappable kernel memory. While no system is 100% secure, the exposure of the DEK is minimal especially when taking into account the DEK is wrapped with a one-time use AES key each time the agent requests the DEK from the Key Management Service. Additionally, due to this approach, changing the DEKs frequently is not warranted and would be

costly. However, customers can define an acceptable use period for the DEKs and change the keys based on that period. Customers can change the DEKs by backing up and restoring the data in the guard points or by using the data transformation utilities. Either approach requires a protection service outage to convert all encrypted data from the old key to the new key.

3.4.1 Block Storage Encryption

BT Cloud Compute offers the block storage services of multiple Cloud vendors to its customers. The block storage services offer the creation and management of virtualized raw block devices of a user-specified size. These block devices are typically used as data volumes by attaching them to customers' virtual machines using the appropriate Cloud service's management API. Currently BT Cloud Compute offers file and volume encryption services on top of Amazon AWS and Citrix CloudStack block storage services.

Amazon Elastic Block Store

Amazon Elastic Block Store (EBS) provides raw block-level storage that can be attached to Amazon EC2 instances. These block devices can then be used like any raw block device. In a typical use case, this would include formatting the device with a filesystem and mounting said filesystem on a virtual machine. The data in the EBS volume or filesystem persists independently from the running life of a virtual machine instance. Multiple data volumes or filesystems can be attached to a virtual machine instance. It is also possible to detach an EBS volume from one virtual machine and attach it to another virtual machine. Amazon EBS also allows to create encrypted EBS volumes, however existing EBS volumes or filesystems cannot be encrypted and the encryption keys are stored and managed by Amazon services.

Citrix CloudStack Primary storage

Citrix CloudStack has a similar concept of block storage in the form of "Primary" storage. Primary storage is associated with a CloudStack cluster, and it stores virtual disks for all the virtual machines running on hosts in that cluster. On KVM and VMware, Primary storage can be provisioned on a per-zone basis. Multiple Primary storage servers can be added to a cluster or zone, but at least one is required. It is typically located close to the hosts for increased performance. CloudStack manages the allocation of guest virtual disks to particular primary storage devices.

Similar block storage services are offered by almost all IaaS Cloud vendors, hence in theory our solution is applicable equally to almost all Cloud block storage service providers. We have picked a public and a private Cloud platform as a sufficient use case to demonstrate the architecture of this component of the DPaaS model.

Architecture of Block Storage Encryption service

The architecture for the block storage encryption component of the DPaaS comprises of three main modules, and is shown in Figure 3.2.

The first module is the BT Service Store that is used to provision and manage the life-cycle of the component's service to the customers or tenants. Each tenant gets a compartmentalized view of the service store and the data protection service, as discussed in the previous sections. The BT Service Store also has the ability to install and configure the data encryption agents on virtual

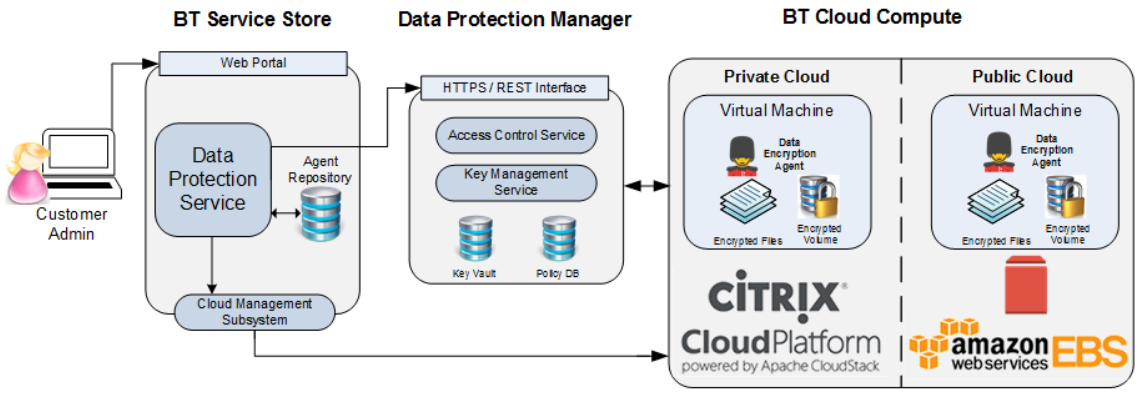

Figure 3.2: Architecture of the block storage encryption service component of the DPaaS solution

machines on different supported Cloud platforms. These agents are stored in a software repository on the BT Service Store.

The second module is the Data Protection Manager, which enables the customers to define the access control and key release policies via the BT Service Store or its own Web/REST interface. The Data Protection Manager also contains secure vaults where the customers can import, export and manage their encryption keys and access control policies. The Data Protection Manager is able to communicate with the data encryption agents running on the virtual machines on different Cloud platforms over secure communication channels like TLS.

The third and last module is the data encryption agent that is provisioned on the target virtual machine by the BT Service Store's Service Orchestrator (SO). After successful provisioning, the agents on the virtual machines use PKI-based authentication to identify themselves to the Key Management Service that is being managed by the Data Protection Manager module. If the agent successfully passes the authentication phase, the Key Management Service issues the keys necessary to encrypt the files and data volumes present on the block storage attached with the virtual machine. After the completion of the encryption process, the access to the protected files and volumes is enforced by the agent as well. Upon receiving a data access request, the agent checks the Access Control Service for the policy associated with the protected file or volume. It submits an encryption key release request to the Key Management Service if the access request is approved by the Access Control Service. Once the Key Management Service issues the encryption key, the agent can use the key to decrypt the data requested.

3.4.2 Object Storage Encryption

BT Cloud Compute offers the object storage services of multiple Cloud vendors to its customers. Object storage refers to an approach of addressing and manipulating discrete units of storage called objects. It offers storage services on a higher level of abstraction than that present on the standard block storage devices like hard disks. Instead of offering the abstraction of a logical array of unrelated blocks in which data can be written or read using an index in the array, as in block storage devices, it offers the abstraction of a flat address space called a storage pool. All objects exist in the same level and there is no concept of hierarchy i.e. one object cannot be placed inside another object. An object itself can be viewed as a container that contains metadata associated with some data. Each object is assigned a globally unique identifier (GUID) which is used by the users to retrieve the object without needing to know the physical location of the data. Object

storage is commonly used in Cloud computing environment in a Write-Once-Read-Many model (WORM) for implementing data backup services.

Amazon S3

Amazon S3 (Simple Storage Service) is an object storage service offered by Amazon Web Services (AWS). Objects are organized into buckets and are identified within each bucket by a unique, user-assigned key. Buckets and objects can be created, listed, and retrieved using either a REST-style HTTP interface or a SOAP interface. Requests are authorized using an access control list associated with each bucket and object. S3 also supports encryption-at-rest of the user data once it has been uploaded with server-side encryption (SSE), where the encryption keys are managed or processed by Amazon.

Caringo Swarm

Caringo Swarm is a symmetric parallel object store that enables massively parallel processing with built-in large bandwidth to overcome the performance limitations of traditional file systems. In addition, a dynamically distributed index used to locate objects is located in RAM in order to further improve performance. Furthermore, the metadata (which includes all system, policy, and custom metadata) is encapsulated with the data within the object. That means that the object is portable and dynamic, even as it is decoupled from specific hardware, location, and applications. It employs erasure coding that uses a variant of the Reed-Solomon code. This ensures that the up-time is high, the risk of data loss is extremely low, and the storage redundancy requirements are less than what RAID would require for the same level of reliability. However, it does not have native encryption support, and does not support integration with encryption agents.

Architecture of Object Storage Encryption service

The architecture for the object storage encryption component of the DPaaS comprises one main module, i.e., the BT Cloud Encryption Gateway, as shown in Figure 3.3.

The gateway solution relies on the Data Protection Manager, described in the previous section, for encryption key and policy management. As a result, customers never need to relinquish control of cryptographic keys to the service provider and data never leaves the virtual machines unencrypted or unaccounted.

The main role of the BT Cloud Encryption Gateway is to act as a proxy that can be used to intercept objects being sent to the object storage services and transparently encrypt them during transfer. The proxy can be deployed as a gateway in either the customers' premises as a forward proxy or in the Cloud environment as a reverse proxy. In addition to the proxy, it also implements a basic key-value store to track the state of the objects being encrypted in the gateway, connectors for the supported Cloud object storage services (S3, Caringo etc.), and the data encryption agent performing the core object encryption and decryption operations.

3.4.3 BT Big Data Services

BT offers various Big Data services to its customers, which enable customers to transform the way they manage and process business data, shorten typical development and deployment cycles and extract the maximum value from their Big Data. In the context of ESCUDO-CLOUD, the specific Big Data technology that has been identified and chosen to be included within the scope

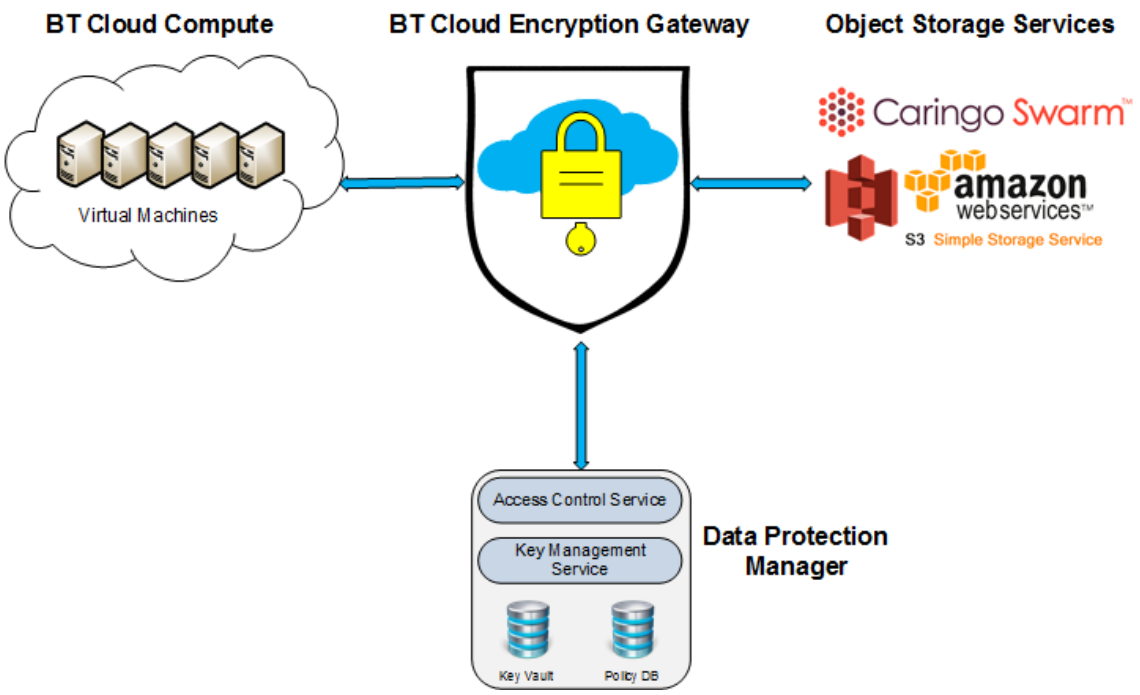

Figure 3.3: Architecture of the object storage encryption service component of the DPaaS solution

of DPaaS is the Hadoop Distributed File System (HDFS) [HDF17]. HDFS is a Java-based fault-tolerant distributed filesystem for storing large volumes of data, with a master/slave architecture. Typically an HDFS cluster consists of a single *NameNode*, a master server that manages the file system namespace and regulates access to files by clients, and multiple *DataNodes*, that manage storage attached to the nodes that they run on.

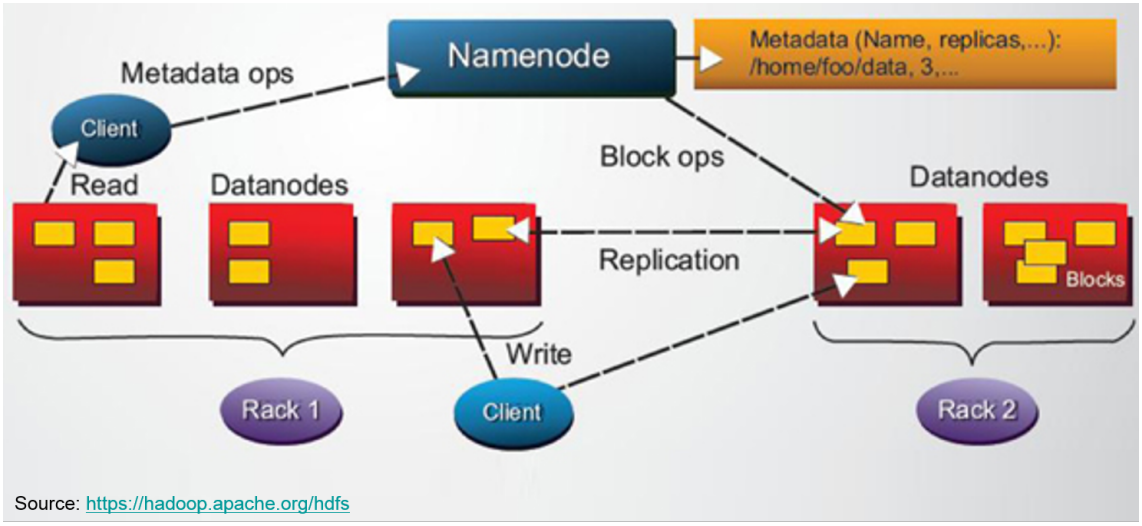

Figure 3.4: HDFS Architecture

In HDFS, the file content is split into large blocks (typically 128 MB), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local filesystem on the DataNodes, e.g., ext3, ext4, XFS etc. The NameNode actively monitors the number of replicas of a block. When a replica of a block is lost due to a DataNode failure or

disk failure, the NameNode creates another replica of the block. The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in memory. However, the NameNode does not directly send requests to DataNodes. It sends instructions to the DataNodes by replying to heartbeats sent by those DataNodes. The instructions include commands to replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, or shut down the node. A simplified logical architecture of HDFS is shown in Figure 3.4. As HDFS exposes a filesystem namespace and allows user data to be stored in files, it is quite similar in logical structure to the Cloud block storage services. Therefore, we aim to re-use some of the work done in the design and development of the block storage encryption component for Big Data storage clusters.

A further motivation for using HDFS as the Big Data storage service is that BT Research and Innovation already offers "Hadoop as a Service" to its customers, which is essentially a Hadoop Cluster abstracted as a shared service platform . This platform is built on top of a 80+ nodes Hadoop cluster optimized for map/reduce jobs (6 TB local disks, 1:1 core:spindle ratio, 8 GB for JVM per map/reduce slot) and deployed in BT data center in Sheffield, UK.

3.4.4 Traditional HDFS Encryption Approaches

Typically, data encryption can be applied at three different levels within an HDFS cluster.

Disk-level encryption

In this approach, typically the whole disk (either virtual or physical) is encrypted, except for a small portion containing the Master Boot Record (MBR), or similar boot-strapping construct that starts the operating system loading sequence. As disk-level encryption generally uses the same key for encrypting the whole disk, all of the data on the disk is decryptable while the system is running.

Therefore, although disk-level encryption is easy to deploy and high performance, it only really protects against physical theft as a run-time attacker has access to all files. Also, it is quite inflexible in the context of encryption granularity.

Filesystem-level encryption

In this approach, data encryption is applied at the Linux filesystem level, which can be thought of as being one layer below the HDFS. Using this encryption method, all files in a volume can be encrypted, i.e., we can encrypt cluster data both inside and outside HDFS, such as temporary files, configuration files and metadata databases. Current implementation of HDFS require one of the three Linux filesystems; ext3, ext4 or XFS.

Although this method and level of encryption offers high performance, application transparency, and is typically easy to deploy, it is unable to model some application-level policies. For example, in multi-tenant environments, a lot of customers want to encrypt their HDFS data based on the end user. This of course is not possible at the level of filesystems as they are at a different level of abstraction than HDFS. Therefore there is a loss of context if the customer wants to perform policy-based data encryption.

HDFS-level encryption

In this approach, data encryption is applied at the HDFS folder level, allowing the customers to encrypt some folders and leave others unencrypted. This approach, also known as "HDFS Transparent Encryption", is end-to-end, i.e., it protects data-at-rest and in transit, which makes it more efficient than implementing a filesystem level encryption approach. Also, only HDFS clients can encrypt or decrypt data and HDFS servers do not have access to unencrypted data or encryption keys. This allows for a separation of duties so that no unauthorized party has unrestricted access to the data and keys. For transparent encryption, it introduces the abstraction of an *encryption zone*. An encryption zone is a special directory whose contents can be transparently encrypted upon write and transparently decrypted upon read. Each encryption zone is associated with a single encryption zone key which is specified when the zone is created. Each file within an encryption zone has its own unique DEK. DEKs are never handled directly by HDFS. Instead, HDFS only ever handles an encrypted DEK (EDEK). Clients decrypt an EDEK, and then use the subsequent DEK to read and write data. HDFS DataNodes simply see a stream of encrypted bytes.

Although this approach provides a level of security against OS and filesystem-level attacks, it has a few drawbacks. Firstly, it cannot encrypt any data outside HDFS, therefore sensitive data in temporary and configuration files etc. can be left unprotected. Secondly, AES-CTR is the only supported encryption algorithm in the current HDFS implementation. This is probably due to concerns about performance overheads due to encryption and decryption operations. However, a lot of customers prefer using AES in CBC mode due to their internal policies and guidelines or due to regulatory compliance requirements.

3.4.5 DPaaS Big Data Encryption Approach

As discussed in the previous section, most of the existing data security solutions in this domain operate on one of these three levels of encryption or combinations thereof. The BT DPaaS solution combines the strengths of all these approaches and can work at all three levels, depending on the customers' data encryption model and enforced encryption and decryption of data based on the key release and access control policies defined by them. By delivering a single data protection solution that offers coverage of all these areas, DPaaS enables its users to leverage centralized controls that optimize efficiency, flexibility and compliance. These centralized controls include fine-grained, policy-based access controls that restrict access to encrypted data by allowing only approved processes and users access to data.

Table 3.1 shows the salient features of the different data protection approaches, with access control granularity and flexibility in ascending order (lower entries have more access control granularity).

Table 3.1: HDFS Encryption Comparison

Encryption Service	Data Type	Data Location	Additional Services Required
Disk Level	Any	Disk	dm-crypt [DMC17] + LUKS [LUK17]
Filesystem Level	Any	Local Filesystem	eCryptfs [eCr17], ext4-crypt [EXT17], EncFS [ENC17], etc.
HDFS TE	HDFS	HDFS	Java Keystore [JAV17] + Hadoop-KMS [KMS17]
BT DPaaS	Any	Disk + Local Filesystem + HDFS	N/A

3.4.6 Architecture of DPaaS Big Data Encryption service

The architecture for the HDFS encryption component of the DPaaS comprises of three main modules, and is shown in Figure 3.5.

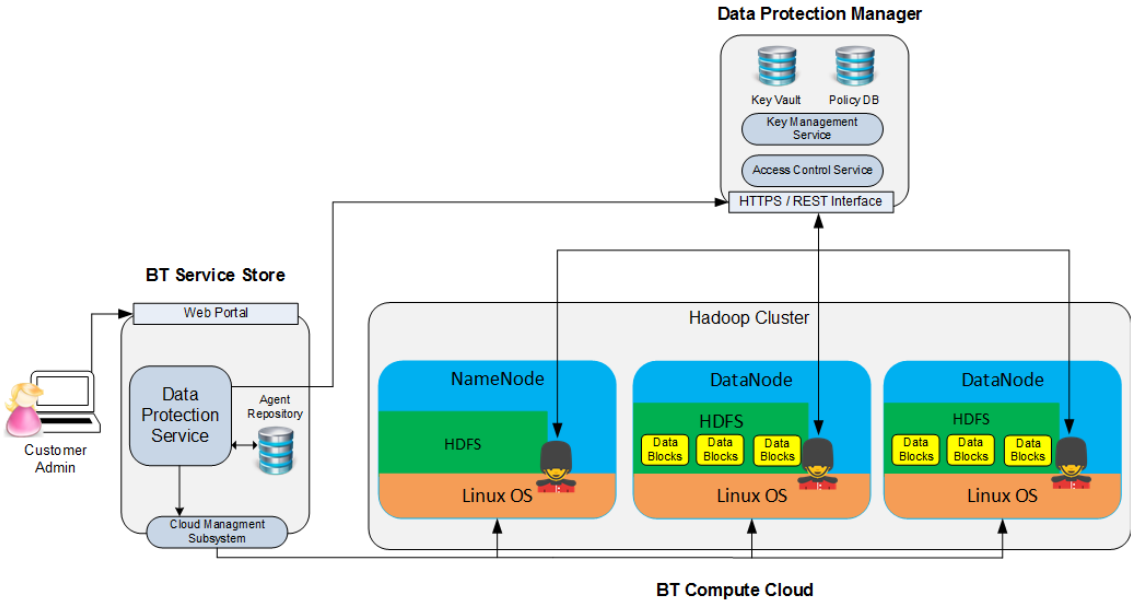

Figure 3.5: Architecture of the HDFS encryption component of the DPaaS solution

The first module is the BT Service Store that is used to provision and manage the life-cycle of the HDFS cluster to the customers or tenants, usually through a Hadoop cluster management and monitoring service like Apache Ambari [AMB17]. Each tenant gets a compartmentalized view of the Service Store and the Data Protection service, as discussed in the previous sections. The BT Service Store also has the ability to install and configure the Data Encryption Agents on HDFS NameNode and DataNodes, on different supported Cloud platforms. These agents are stored in a software repository on the BT Service Store.

The second module is the Data Protection Manager, which enables the customers to define the access control and key release policies via the BT Service Store or its own Web/REST interface. The Data Protection Manager also contains secure vaults where the customers can import, export and manage their encryption keys and access control policies. The Data Protection Manager is able to communicate with the Data Encryption Agents running on the NameNode and DataNodes of the customer's HDFS cluster, over secure communication channels like SSL/TLS.

The third and last module is the Data Encryption agents that are provisioned on the target virtual machines, hosts of the customer's HDFS cluster, by the BT Service Store's SO. After successful provisioning, the agents on the HDFS cluster nodes use PKI-based authentication to identify themselves to the KMS that is being managed by the Data Protection Manager module. If the agents successfully pass the authentication phase, the KMS issues them with the DEKs necessary to encrypt the data blocks stored on the HDFS storage. After the completion of the encryption process, the access to the protected data blocks is enforced by the agents as well. Upon a data access request, the agents check the Access Control Service for the policy associated with the protected data blocks. It submits an encryption key release request to the KMS, if the access request is approved by the Access Control Service. Once the KMS issues the DEK, the agents can use it to decrypt the data requested.

3.5 Prototype Implementations

In this section, we describe in some detail the implementation of the prototypes for the block storage encryption service, the object storage encryption service, and the Big Data encryption service of the DPaaS solution developed and integrated by BT under the scope of ESCUDO-CLOUD Use Case 3.

3.5.1 Block Storage Encryption

The detailed architecture of the block storage encryption service of DPaaS is shown in Figure 3.6. In subsequent parts of this section, we respectively explain the reference design of the prototype service from two different points of view: component view and database view.

Component View

The block storage encryption service of DPaaS consists of three main components: Tenant Management, Cloud Platform Management and Security Solution Management.

Tenant Management (TM): This is the main component of the framework. The TM component is in charge of managing customers registering to use the data protection service. In addition, it maintains information of the user Cloud platform and security solutions that can be embedded in the data protection service.

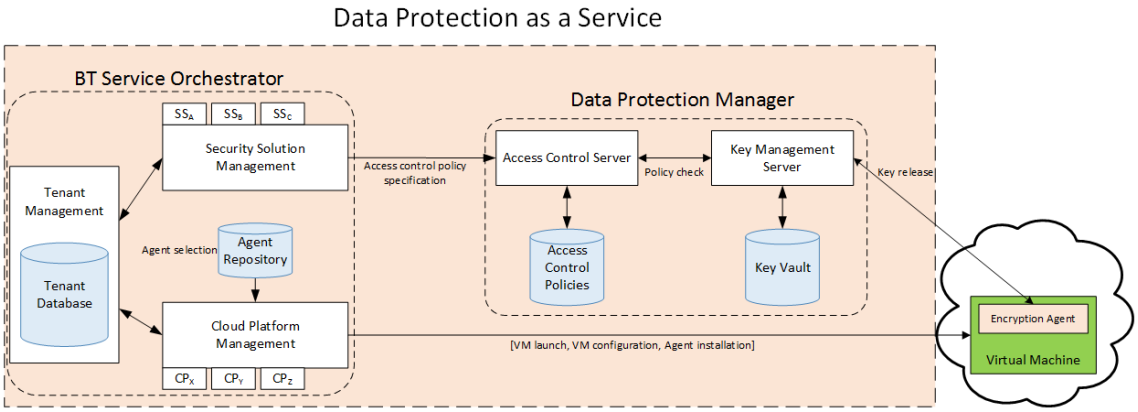

Figure 3.6: Implementation reference of the block storage encryption service prototype

Cloud Platform Management (CPM): To support a Multi-Cloud environment, this component provides an interface consisting of APIs for communications with the supported Cloud platforms. For each Cloud platform the framework supports, a Cloud management plug-in is implemented and attached to this component. Among APIs defined in the interface, some of them are mandatory while others are optional for implementation. For example, it is required to implement the APIs that connect to the Cloud platform for virtual machine deployment and virtual machine termination because these operations are involved in the security agent installation and uninstallation actions.

Security Solution Management (SSM): Similar to the CPM component, to support different security solutions, the SSM component defines an interface with APIs for communications with the security solution servers and for each security solution the framework supports, a security plug-in is needed. Basic APIs in this interface include access control policy definition and data encryption/decryption requests.

As shown in Figure 3.6, while the TM component interacts with both the CPM and the SSM components, the CPM and SSM components are independent of each other. Besides, even though the TM component requires interaction with the CPM and SSM components, this interaction is loosely coupled. Basically, based on the registered information of the users and depending on the specific requests of users, the TM component will directly trigger the corresponding plug-ins inside the CPM and SSM components. For example, if a user chooses to protect data in a virtual machine deployed in the Amazon EC2 platform and he chooses to employ a security solution from Trend Micro, the Amazon EC2 plug-in and Trend Micro plug-in will be called by the TM component. Next time, if the user chooses to protect data in a virtual machine deployed in the CloudStack platform and he chooses to employ a security solution from SafeNet, the CloudStack plug-in and SafeNet plug-in will be called by the TM component. This design provides the scalability for the framework to support several Cloud platforms and security solutions.

Database View

The framework is supported by four databases: the tenant database, the access control policies database, key store/vault, and the agent repository. Among these four databases, only the first one, which is the tenant database, is located inside the BT Service Store framework. The other two databases are situated outside the Service Store framework and have separate interfaces for policy management and key management. In this way, fine-grained access control policies can be defined and users can customize the policies via the external interfaces without going through the Service Store framework.

However, by having these two databases outside the Service Store framework, we need to also define interfaces to connect them with the SSM component. At the moment, we employ a simple REST interface to set-up default basic access control policies for users and leave users the freedom to add-in or modify the policies later through the external interfaces.

3.5.2 Object Storage Encryption

The detailed architecture of the object storage encryption service of DPaaS is shown in Figure 3.7. It uses most of the same components as the implementation described in the previous section. The only main addition in this case is the BT Cloud Encryption Gateway. The Cloud Encryption Gateway can be deployed in either the customers' premises as a forward proxy or in the Cloud environment as a reverse proxy, and implements the following main functions:

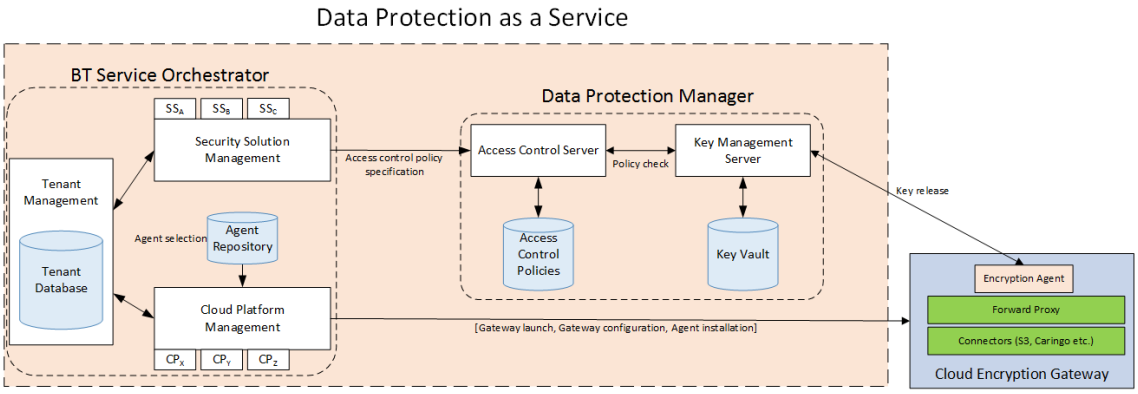

Figure 3.7: Implementation reference of the object storage encryption service prototype

Forward Proxy

The main feature of the Cloud Encryption Gateway is to act as the proxy server for all the object storage requests and responses coming to and from the S3 and Caringo object storage services. This proxy server intercepts these objects being sent to the object storage services and transparently encrypts them during the caching phase. In addition to the basic proxy functionality, we also implement a basic key-value store using MongoDB to keep track of the state of the objects being encrypted in the gateway.

Object Storage Connectors

Similar to the CPM component described in the block storage encryption implementation, in the object storage encryption solution we made use of different object storage connectors that provide the interfaces to interact with different Cloud object storage services. So far we have made use of Amazon S3 and Caringo Swarm connectors, to which we provide the appropriate customer's object services' credentials when the Cloud Encryption Gateway has been set-up and configured by the Service Store. The proxy server uses these connectors to upload and download objects from the respective object store.

3.5.3 Big Data Encryption

The detailed architecture of the Big Data encryption service of DPaaS is shown in Figure 3.8. It uses most of the same components as the implementation described in the previous section. As in the block storage encryption implementation, the CPM component is responsible for launching and configuring the base VMs, which will constitute the cluster nodes of the HDFS cluster, on the selected Cloud platform. It also provisions and configures the Data Encryption Agents on all the cluster nodes, be they either NameNodes or DataNodes.

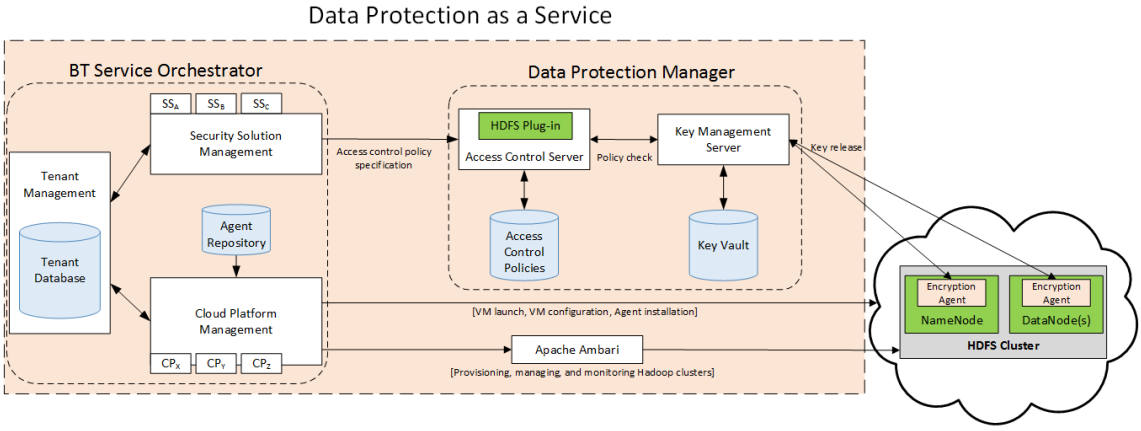

Figure 3.8: Implementation reference of the Big Data encryption service prototype

Apache Ambari

The main difference between this reference implementation and the block storage encryption implementation is the use of the Apache Ambari service to deploy and configure relevant Hadoop services on the cluster, as chosen by the customer organization. The Ambari service is also used to perform part of the encryption agents' configuration so that they are able to understand the HDFS

virtualization and perform encryption/decryption and policy enforcement operations at the HDFS level.

HDFS Plug-in

Similar to the CPM component described in the object storage encryption implementation, in the Big Data encryption solution we make use of the HDFS Plug-in that provides the capability and interface to define encryption policies for the HDFS files and directories stored in HDFS cluster's DataNodes. Due to this component, the customers can even selectively encrypt HDFS folders with different keys, providing multi-tenancy support. Additionally, the customers can also define user-based access control rules for HDFS files in their HDFS cluster.

3.6 Integration between BT DPaaS and EncSwift

In Year 3 of ESCUDO-CLOUD we also investigated the possibility of integration between the BT DPaaS and the EncSwift solution [BDF+16] developed by UNIBG as part of the work done in WP 2. EncSwift is a tool for protecting data confidentiality in OpenStack Swift already introduced in [PBD+16] and [FR17], that uses different encryption keys to protect resources, in accordance with fine-grained policies defining access to those resources. In dynamic systems with frequent policy updates, the use of encryption brings non-negligible complexity since keys must be shared and revoked in an efficient and trusted way.

In this context, the integration of the BT DPaaS, more specifically its Key Management Service, was carried out with EncSwift, in a scenario with frequent policy updates, and thus with the generation and administration of a large number of encryption keys. The main outcome of this effort was the implementation of a solution for protecting data stored at-rest in OpenStack Swift based object storage service, described in detail in Deliverable 2.6 of the ESCUDO-CLOUD project. This collaborative effort also resulted in a publication [BRS17].

3.7 Summary

This chapter showcases the detailed design process of the DPaaS solution. The current design describes the scope and granularity of some of the main components of the solution, especially those related to offering data protection for Cloud block, object and Big Data storage services. This helps us in moving towards the final solution design for this task. The main focus of the design of the DPaaS solution is the customer ownership and control of the outsourced data and its protection. This is achieved due to the integration of key management and access control features, as only the customers exercise the control over the encryption keys. Only they have the ability to regulate access to their data through policy based enforcement of access control attributes. They will also have the assurance that their data will remain protected from the untrusted Cloud service providers.

As the customers are provided with the choice of selecting storage services from multiple Cloud service providers, by retaining control of key management the customers have more flexibility in meeting regulatory and privacy requirements and ensuring data confidentiality and secure access. Thus, by utilizing the BT Service Store and its features, the Data Protection as a Service is able to support different security solutions in a Multi-Cloud environment. The BT Service Store

provides full automation of data encryption services to users as a complete life-cycle, from data encryption to data decryption.

4. Conclusion

The chapters of this deliverable provide the M1-M34 overview of the conducted WP4 tasks. As a summary, the thematic coverage was the following.

Security metrics and assessment. In D4.1, we addressed trust metrics and especially the SLA-based solutions for enabling users to express and reason about the trust associated with different providers for matching their requirements. Furthermore, we developed two evaluation techniques for conducting the quantitative assessment and analysis of the SLA based security level provided by the CSPs. Finally, we validated the two evaluation techniques using two use case scenarios and a prototype, leveraging actual real-world CSP SLA data derived from the publicly available CSA STAR.

In D4.2, we presented a novel SLA-based service selection methodology for the Multi-Cloud environment that (a) provided a MCSLA construction, and (b) evaluated the service levels provided in Multi-Cloud services, considering the dependencies among the services provided by its participants.

Multiple providers for security. We have extended the shuffle index approach, originally developed in Work Package 2, to the multiple providers scenario. In particular, we have distributed the shuffle index among different providers to enhance the security guarantees it offers (D4.2). We have implemented our approach on different Cloud providers (D4.3), and deployed it on a real industrial scenario, proving its good performance. We have also investigated the use of distributed Cloud storage and All-or-Nothing-Transform encryption mode to guarantee security. Lastly, we have proposed an approach for users to specify their constraints and preferences on the characteristics of Cloud services available on the market, whose enforcement allows users to rely on those services that satisfy their needs.

Data Protection as a Service for Federated Cloud Storage. We cover the design and development of the DPaaS solution developed and integrated by BT as the main outcome of T4.3 of the ESCUDO-CLOUD project. We have listed and described tools and techniques identified and selected for the DPaaS solution. The solution guarantees the protection of the ownership of customer data in a Multi-Cloud environment by encrypting the data on the Cloud and associating its access control with the key release process for the decryption of the data. The keys are always under the control and management of the customers or an authorized third party, and access is only granted if the request fulfils the constraints of customer-defined policies. Due to the variety of the Cloud storage services that can be utilized by the customers to store data on different mediums, the task also defined the scope and gave reference architectures of components that targeted a particular type of Cloud storage service, i.e., Cloud-based block storage, object storage (D4.2) and Big Data services (D4.4).

Bibliography

[AMB17] Apache Ambari. *https://ambari.apache.org/*, 2017.

[BBB+17] E. Bacis, A. Barnett, A. Byrne, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Distributed shuffle index: Analysis and implementation in an industrial testbed. In *Proc. of CNS*, pages 35–36, Las Vegas, NV, USA, October 2017.

[BDF+16] E. Bacis, S. De Capitani di Vimercati, S. Foresti, D. Guttadoro, S. Paraboschi, M. Rosa, P. Samarati, and A. Saullo. Managing data sharing in OpenStack Swift with Over-Encryption. In *Proc. of the 3rd ACM Workshop on Information Sharing and Collaborative Security (WISCS 2016)*, Vienna, Austria, October 2016.

[BDF+17] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Distributed shuffle index in the cloud: Implementation and evaluation. In *Proc. of IEEE CSCloud*, pages 139–144, New York, USA, June 2017.

[BdVF+17] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Final versions of tools for security with multiple providers. Deliverable D4.3, ESCUDO-CLOUD, June 2017.

[BRS17] E. Bacis, M. Rosa, and A. Sajjad. EncSwift and Key Management: An Integrated Approach in an Industrial Setting. In *IEEE Conference on Communications and Network Security*, pages 492–495, Las Vegas, Nevada, USA, October 2017.

[CPRT05] V. Casola, R. Preziosi, M. Rak, and L. Troiano. A Reference Model for Security Level Evaluation: Policy and Fuzzy Techniques. *Journal of Universal Computer Science (UCS)*, 11(1):150 – 174, 2005.

[DDD+16] G. Ducatel, J. Daniel, T. Dimitrakos, F. El-Moussa, R. Rowlingson, and A. Sajjad. Managed security service distribution model. In *4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 404–408. IEEE, 2016.

[DFP+11] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Efficient and private access to outsourced data. In *Proc. of ICDCS*, pages 710–719, Minneapolis, Minnesota, USA, June 2011.

[DFP+15] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Shuffle index: Efficient and private access to outsourced data. *ACM Transactions on Storage (TOS)*, 11(4):1–55, October 2015. Article 19.

[DFP+17] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Three-server swapping for access confidentiality. *IEEE TCC*, 2017. to appear.

[DLP16] S. De Capitani di Vimercati, G. Livraga, and V. Piuri. Application requirements with preferences in cloud-based information processing. In *Proc. of IEEE RTSI 2016*, pages 1–4, Bologna, Italy, September 2016.

[DMC17] dm-crypt. *https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt*, 2017.

[DMM+12] I. Drago, M. Mellia, M. Munafò, A. Sperotto, R. Sadre, and A. Pras. Inside dropbox: Understanding personal cloud storage services. In *Proc. of Internet measurement conference*, pages 481–494, 2012.

[DvDF+16] S. Devadas, M. van Dijk, C.W. Fletcher, L. Ren, E. Shi, and D. Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Proc. of TCC*, pages 145–174, TelAviv,Israel, January 2016.

[eCr17] eCryptfs - The enterprise cryptographic filesystem for Linux. *http://ecryptfs.org/*, 2017.

[ENC17] EncFS: an Encrypted Filesystem for FUSE. *https://github.com/vgough/encfs*, 2017.

[EXT17] ext4-crypt. *https://github.com/gdelugre/ext4-crypt*, 2017.

[FR17] S. Foresti and M. Rosa. Final tools for the protection of integrity and confidentiality of data and access. Deliverable D2.5, ESCUDO-CLOUD, March 2017.

[HDF17] HDFS Architecture. *https://hortonworks.com/apache/hdfs*, 2017.

[Int16] International Organization for Standardization ISO/IEC 19086. Information Technology - Cloud Computing - Service Level Agreement (SLA) Framework and Terminology. Online:https://www.iso.org/standard/67545.html, 2016.

[JAV17] Java Keystore. *http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html*, 2017.

[KMS17] Hadoop Key Management Server. *https://hadoop.apache.org/docs/current3/hadoop-kms/index.html*, 2017.

[LLS12] J. Luna, R. Langenberg, and N. Suri. Benchmarking Cloud Security Level Agreements Using Quantitative Policy Trees. In *Proc. of the ACM Cloud Computing Security Workshop (CCSW)*, pages 103–112, 2012.

[Loo11] J. Loope. *Managing Infrastructure with Puppet: Configuration Management at Scale*. O'Reilly Media, Inc., 2011.

[LTTS15] J. Luna, A. Taha, R. Trapero, and N. Suri. Quantitative reasoning about cloud security using service level agreements. *IEEE Trans. Cloud Computing*, 5(3):457–471, 2015.

[LUK17] Linux Unified Key Setup. *https://gitlab.com/cryptsetup/cryptsetup/wikis/LUKS-standard/on-disk-format.pdf*, 2017.

[MTT+16] J. Modic, R. Trapero, A. Taha, J. Luna, M. Stopar, and N. Suri. Novel efficient techniques for real-time cloud security assessment. *Computers & Security*, 62:1–18, 2016.

[PBD+16] S. Paraboschi, E. Bacis, S. De Capitani di Vimercati, S. Foresti, D. Guttadoro, S. Mutti, M. Rosa, and A. Saullo. Tools for protecting confidentiality and integrity of data and access. Deliverable D2.2, ESCUDO-CLOUD, June 2016.

[Saa90] T. Saaty. How to make a decision: the analytic hierarchy process. *European Journal of Operational Research*, pages 9–26, 1990.

[SS13] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *Proc. of CCS*, pages 247–258, Berlin, Germany, November 2013.

[STA11] CSA STAR. Cloud Security Alliance Security, Trust & Assurance Registry. Online: https://cloudsecurityalliance.org/star/, 2011.

[SvS+13] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: An extremely simple Oblivious RAM protocol. In *Proc. of CCS*, pages 299–310, Berlin, Germany, November 2013.

[TMS17a] A. Taha, S. Manzoor, and N. Suri. SLA-based Service Selection for Multi-Cloud Environments. In *Proc. of EDGE Computing (EDGE)*, pages 65–72, 2017.

[TMS+17b] R. Trapero, J. Modic, M. Stopar, A. Taha, and N. Suri. A novel approach to manage cloud security SLA incidents. *Future Generation Comp. Syst.*, 72:193–205, 2017.

[TMT+16] A. Taha, P. Metzler, R. Trapero, J. Luna, and N. Suri. Identifying and utilizing dependencies across cloud security services. In *Proc. of Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 329–340, 2016.

[TTLS14] A. Taha, R. Trapero, J. Luna, and N. Suri. AHP-Based Quantitative Approach for Assessing and Comparing Cloud Security. In *Proc. of Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 284–291, 2014.

[TTLS17] A. Taha, R. Trapero, J. Luna, and N. Suri. A Framework for Ranking Cloud Security Services. In *Proc. of Services Computing (SCC)*, pages 322–329, 2017.

[Zel82] M. Zeleny. *Multiple Criteria Decision Making*. McGraw Hill, 1982.