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

D4.2
First Report on Multi Cloud and Federated Cloud

Editors: Ahmed Taha (TUD)
Ruben Trapero (TUD)
Neeraj Suri (TUD)
Sara Foresti (UNIMI)
Ali Sajjad (BT)
Reviewers: David Bowden (EMC)
Björn Tackmann (IBM)

Abstract

Multi-provider based Cloud services allow users to leverage new services that enhance security, reduce costs and increase availability, among several other benefits. However, the presence of multiple providers, offering multiple types of storage services, introduces new considerations for secure Cloud storage. From the user's point of view, the combination of multiple providers now entails the need to measure and compare the security guarantees provided by the varied providers that integrate capabilities as part of a Multi-Cloud service, along with their different implementations of security in terms of confidentiality, integrity or compliance with Service Level Agreements (SLAs). The Multi-Cloud services also provide the users with innovative features related to the data storage such as the ability to split data and accesses across different providers, which in fact require additional efforts to evaluate the threat of conflicts with the integrity of the data stored in every provider. Alternately, the Federated-Cloud providers can solve such issues by managing secure Cloud storage among providers that can inter-operate or can be centrally managed via a consistent interface, while still working as autonomous entities. This deliverable addresses reasoning about SLA-based security metric dependencies in Multi-Cloud environments, extends techniques for protecting access and pattern confidentiality to the Multi-Cloud scenario, and provides techniques and solutions for protecting data in Federated-Cloud storage scenarios.

Type	Identifier	Dissemination	Date
Deliverable	D4.2	Public	2016.12.30

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	2016.11.30	Initial Release
0.2	2016.12.14	Second Release
1.0	2016.12.30	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	Ruben Trapero (TUD), Ahmed Taha (TUD), Neeraj Suri (TUD), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Ali Sajjad (BT), Rob Rowlingson (BT), Mark Shackleton (BT)
Chapter 1: Introduction	Ruben Trapero (TUD), Ahmed Taha (TUD), Neeraj Suri (TUD), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Ali Sajjad (BT), Rob Rowlingson (BT), Mark Shackleton (BT)
Chapter 2: SLA Based Metrics and Assessment in Multi-Cloud	Ahmed Taha (TUD), Ruben Trapero (TUD), Neeraj Suri (TUD)
Chapter 3: Data Distribution and Swapping for Access Confidentiality	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 4: Data Protection as a Service for Federated Cloud Storage	Ali Sajjad (BT), Rob Rowlingson (BT), Mark Shackleton (BT)
Chapter 5: Conclusions	Ahmed Taha (TUD), Ruben Trapero (TUD), Neeraj Suri (TUD), Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Ali Sajjad (BT), Rob Rowlingson (BT), Mark Shackleton (BT)

Contents

List of Figures

List of Tables

Executive Summary

The benefits of moving data storage to the Cloud are apparent: higher availability, increased storage space, ubiquitous and concurrent data accesses, to name just a few. These benefits often get enhanced when multiple providers coalesce to form either an integrated Multi-Cloud environment or a loosely-coupled Federated-Cloud for enhanced provisioning of (complex) services. The multi-provider Cloud services permit the users to take advantage of a variety of new resources and services from diverse providers. For example, users can now choose across providers with better cost versus services offerings. Performance can be improved as the workload can be balanced among the multiple Cloud providers. Availability can also be improved as the multi-provider Cloud services can better implement data redundancy, distributing several copies of some parts of the data among different providers. However, while the benefits are easy to list, there are also several new security issues that need to be considered. As the data management is distributed among different providers, the users can easily lose control of who is actually managing their data (e.g., Cloud providers may collude and exchange data) and what accesses are transpiring on them. In addition, different providers commonly offer services using different technical solutions that introduce different vulnerabilities. As a result, multi-provider Cloud services must provide the corresponding protection techniques adapted to the Multi-Cloud or Federated-Cloud service, and also related guidance to the providers that are part of the Multi/Federated-Cloud service.
This deliverable explores different protection techniques for multi-provider Cloud services from three perspectives.

- *The evaluation of the service levels provided in Multi-Cloud services.* Different providers might implement different metrics that might entail conflicts between them. This deliverable evaluates dependencies between metrics and uses these dependencies to develop techniques for the accurate assessment of Multi-Cloud services.

- *The definition of techniques based on data distribution and swapping for access confidentiality.* The availability of multiple Cloud providers helps to enhance the protection guarantees offered to data owners. This deliverable extends a technique studied in Work Package 2 to protect access and pattern confidentiality to the Multi-Cloud scenario. The proposed approach exhibits protection guarantees, also in case of collusion among providers.

- *The design, development and integration of tools and techniques that offer the data protection solution for block and object storage services on multiple Cloud platforms.* This deliverable details how the BT Data Protection as a Service solution is built on top of underlying services of key management and access control, and is governed by customer controlled key release policies. The service is offered through a Cloud service store where the customer is able to specify what kind of data protection it wants over which Cloud platform.

1. Introduction

The benefits of a Cloud market composed of multiple different providers are apparent, for example varied services offered at different prices, enhanced data availability with geographical spread, reduced access time and many others. In theory, the security guarantees are also enhanced. However, the reality is that the variety of providers introduces uncertainty regarding data protection and also opens several security concerns such as the loss of control (Which providers are accessing my data?), exposure to different threats (Do the different implementations by different providers imply varied exposure to the same vulnerabilities or open new vulnerabilities) or lack of assurance (Are these providers, individually or collectively, fulfilling the user requirements?).

This deliverable tackles these problems by evaluating the security and data protection from the following three perspectives.

- The evaluation of the service levels provided by a Multi-Cloud, considering the dependencies among the services provided by its participants.
- The definition of techniques based on data distribution and swapping for protecting access confidentiality.
- The design, development and integration of protection techniques in the federated Cloud storage scenario.

Each of these elements is outlined below.

Multi-Cloud security dependencies. Security concerns about Cloud services increase when considering Multi-Cloud services. Although multiple Cloud providers offer similar security services (e.g., encryption key management), they might provide different technical solutions to deal with security threats associated with these security services. Additionally, from the user's point of view, this variety of security services across providers implies different capabilities and prices. Comparing and assessing the offered services by different providers is essential and particularly challenging in the case of compositions of multiple Cloud providers. This is due to the presence of explicit and implicit dependencies across related services, as the users need to have: *i)* awareness about these dependencies when specifying their requirements (something that is not easy given the complexity of these dependencies and considering that users often have limited technical skills to meaningfully parse such information); *ii)* means to know the level of service provided by a Multi-Cloud service.
In this deliverable we present a framework that addresses this problem by: *i)* detecting conflicts resulting from inconsistent customer requirements; *ii)* providing means to resolve the detected conflicts; *iii)* evaluate the Service Level Agreement (SLA[1]) based service level provided by the

[1] An SLA is a formal contract between the CSP and the customer in which Cloud services and the corresponding service level objectives that the CSP agrees to meet are defined.

various Cloud providers (along with the dependencies between the services implemented by each provider).

Data distribution and swapping for protecting access confidentiality. In this deliverable, we build on the results of Work Package 2 and extend protection techniques designed to operate with a single provider to operate with a set of Cloud providers, increasing the protection guarantees they provide to the users. The design of such techniques takes into account the risk of collusion among Cloud providers. This deliverable specifically focuses on the protection, not only of content confidentiality, but also of accesses and patterns of accesses. As already discussed in Work Package 2, protecting the confidentiality of sensitive data content is not sufficient in most scenarios. Indeed, revealing the target of accesses may still reveal sensitive information about both the user accessing the data and the data content itself. Different solutions have been proposed to protect access confidentiality (e.g., PIR, ORAM-based approaches, shuffle index). These solutions consider a scenario characterized by the presence of a single Cloud provider storing the whole dataset. In this deliverable, we illustrate an enhancement of the *shuffle index*, described in Deliverable D2.1 [PFL15], that relies on the presence of three independent Cloud providers to improve protection guarantees. The developed solution has been designed to resist to collusion even among all the providers. We analyze the security guarantees provided by the distributed shuffle index, considering also scenarios where Cloud providers collude.

Data Protection as a Service for Federated-Cloud storage. In this deliverable, we detail the first version of design and implementation of BT's "Data Protection as a Service" (DPaaS) solution. The overall goal of this service is to offer data protection capability for multiple Cloud platforms on different storage media and services. Within the scope of ESCUDO-CLOUD, BT is focusing on data protection over three kinds of Cloud storage services; block storage, object storage and Big Data storage services. The block storage service offers the creation and management of files and virtualized raw block devices of a user-specified size. These block devices are typically used as hard disks by attaching them to the customers' virtual machines using the Cloud service's API. The object storage service offers the storage and management of data as objects, instead of files and blocks. An object is typically comprised of the data, its metadata, and a globally unique identifier. The Big Data storage services are offered in the form of Hadoop File System (HDFS) clusters. However, the initial version of this solution within the scope of D4.2 will provide data protection solution for file and block storage services on multiple Cloud platforms, as well as Object storage services. The DPaaS will be built on top of underlying services of key management and access control based on customer-controlled key release policies. The service will be offered through a Cloud service store where the customer organization is able to specify what kind of data protection it wants for its end-users. The service store will provide the interface for the customers of the DPaaS to access and manage these storage services. The service store also has the ability to install and configure the encryption agent software on virtual machines on multiple Cloud platforms. As a result of this structure, the core solution architecture will be a common encryption, key management and access control system but will have different abstraction layers to cater for the underlying storage media.

1.1 Interactions with other Work Packages

The work discussed in this document is related to and interacts with the other ESCUDO-CLOUD work packages as follows (Figure 1.1):

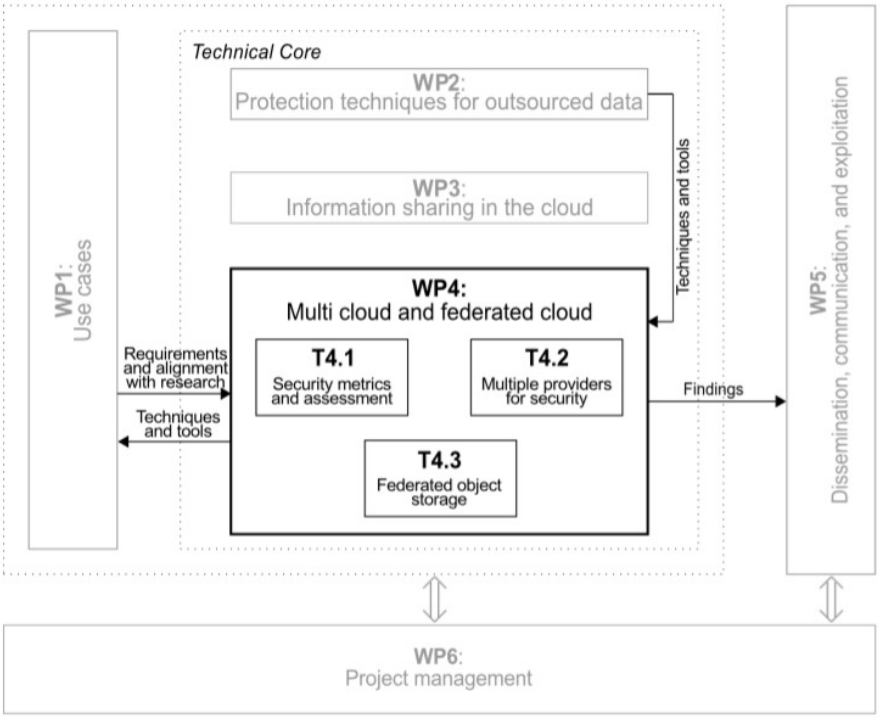


Figure 1.1: Interactions of D4.2 with other Work Packages of ESCUDO-CLOUD

- **Work Package 1** provides the validation use cases for ESCUDO. The use cases and related requirements from deliverables D1.1 and D1.2 form the basis that the protection techniques for Multi-Cloud services provided in this document are intended to address. In turn, the techniques discussed in this report will be applied to the use cases as part of Work Package 1 and the results will be communicated in deliverables D1.3, D1.4 and Work Package 1.2.
- **Work Package 2** provides the design and implementation of protection techniques for outsourced data. The techniques and tools provided by Work Package 2 are considered as part of the applicability discussion in this document with respect to its applicability to Multi-Cloud environments. In fact, we directly build on the shuffle-index approach studied in Work Package 2, and we extend it to operate with multiple Cloud providers to provide better access and pattern confidentiality guarantees.
- **Work Package 5** receives the protection techniques for Multi-Cloud discussed in this document for the purposes of dissemination and exploitation.

1.2 Outline

The remainder of this document is structured as follows. Chapter 2 describes the assessment techniques for Multi-Cloud. Chapter 3 describes a solution based on data distribution and swapping for protecting access confidentiality. Chapter 4 describes the protection techniques for the Federated-Cloud storage scenario. Chapter 5 summarizes the contributions of this deliverable.

2. SLA Based Metrics and Assessment in Multi-Cloud

An ongoing development in Cloud computing targets the combination of resources and services from multiple Clouds. Improving the quality of services, while optimizing service cost; the ability to migrate among several providers; avoiding vendor lock-in; and the need of particular Cloud services which are not provided elsewhere are some of the reasons for using services from multiple Clouds. Currently, two basic delivery models exist for multiple Clouds: Federated-Cloud and Multi-Cloud (cf., Section 2.1.1). Each model defines its own degree of collaboration between the involved Cloud Service Providers (CSPs) and the way the CSP interacts with the Cloud Service Customers (CSCs). In the Multi-Cloud model, the services offered by multiple independent CSPs are aggregated as one or more composite services. We focus on the Multi-Cloud model for its applicability to the ESCUDO-CLOUD use cases, and as it gives customers the freedom to select the Cloud providers that best satisfy their requirements. This is achieved through a third-party (termed as Multi-Cloud middleware in this deliverable) that is responsible to deal with the variations of Cloud provider APIs [Far14].

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[1] across them. For example, Dropbox depends on Amazon's S3 service for storage and on EC2 for computation [DMM+12]. This example shows a SaaS provider that depends on other IaaS providers to support its software.

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 customer 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 customer may require an unachievable level of a dependent service which can result in the customer requirements to be impossible to satisfy. The question remains as to how customers can (a) score and then select services for each single requirement, and (b) on 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 customer.

[1] 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.

This chapter is organized as follows. Sections 2.1.1 and 2.1.2 develop the background and the basic terminology related to the Multi-Cloud environment and the service dependencies respectively. The State of the Art for SLA-based solutions in the Multi-Cloud environment is presented in Section 2.1.3, and Section 2.2 highlights the innovations contributed by ESCUDO-CLOUD. This is followed by a description of the novel ESCUDO-CLOUD quantitative service-level assessment framework in Section 2.3 that considers service dependencies at the level of SLAs. Section 2.4 presents the real-world use-cases that validate the Multi-Cloud service evaluation and composition as well as dependency management.

2.1 State of the Art

2.1.1 Multiple Clouds Delivery Models

The NIST report [NIS11] outlines that multiple Clouds can be used serially, when an application or service is moved from one Cloud to another, or simultaneously, when services from different Clouds are used. The reasons for selecting services and resources from multiple Clouds are various and have been reported in several research/practitioner publications (such as [PWF11, SBKA13]). Currently, there are two basic delivery models in place for multiple Clouds. The first is the Federated Cloud model where the CSPs establish agreements with each other in order to enhance the service offer to their service consumers. The alternate is the Multi-Cloud where, unlike a federation of Clouds, the Multi-Cloud model does not imply interconnection and sharing of providers' infrastructures. The term Multi-Cloud denotes that a third party is building unique entry points for multiple Clouds, without a prior agreement with and between the CSPs [Pet14]. The CSPs provide a composite Cloud service as set of services provisioned by different CSPs and aggregated as one service by the third party. Each service offered by a provider is specified using the provider SLA, as shown in Figure 2.1. In the Multi-Cloud model, the third party contracts the CSPs, negotiates the terms of the services required by the customer, as offered by the CSPs, and subsequently monitors the fulfillment of the SLAs. The third party can be a service on behalf of the Cloud customer[2] or the customer uses library-based software for managing resource provisioning and scheduling[3]. According to [GB14], the Cloud broker[4] is often part of the service or library. Services are defined in [GB14] as an application provisioning that is carried out by a service that can be hosted either externally or in-house by the Cloud customers. Most such services include broker components in themselves. Typically, application developers specify an SLA or a set of provisioning rules, and the service performs the deployment and execution in the background, in a way respecting these predefined attributes. Furthermore, libraries are defined as custom application brokers that directly take care of provisioning and scheduling application components across Clouds [GB14].
It is important to note that variations between the current application programming interfaces (APIs) often hinder the easy composition or configuration of services to be consumed from multiple Clouds. In order to use services from multiple Clouds in a real-world scenarios, several additional technical barriers also need to be addressed such as interoperability, portability, data and services mobility, and middleware openness. In order to control the interface between the customer and different providers in a Multi-Cloud scenario, a contractual basis in the form of an

[2] Several deployable services are the results of open-source projects such as mOSAIC, CLOUD4SOA, OPTIMIS.

[3] The most known library-based approaches for managing resource provisioning and scheduling are jclouds, libcloud, δ-cloud and simple Cloud.

[4] An entity that manages the use, performance and delivery of Cloud services and intermediates the relationships between CSPs and customer [Pet14].

SLA, as a composite SLA across different providers, is needed. This composite SLA (named Multi-Cloud SLA and abbreviated as MCSLA in this chapter) contains all contractual services of all involved SLAs and in addition the dependencies that exist between the different aspects of atomic and composite SLAs[5]. In this environment, we focus on the decision-making problem of the selection and deployment of composite SLAs while considering (a) the customer requirements, and (b) handling the dependencies between different Service Level Objective (SLOs[6]).

It is worth mentioning that the research on Multi-Cloud orchestration is still in a very early stage,

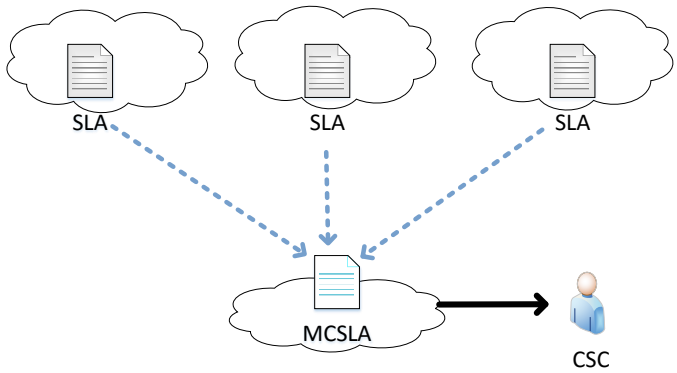


Figure 2.1: Multi-Cloud architecture

with only some exploratory research initiatives appearing from academia and industry. Bernsmed et al. [BJMU11] presented a method for managing the SLA life cycle in the context of federated Cloud services. However, they did not further elaborate on the techniques needed to conduct benchmarking. Jrad et al. [JTS12] proposed an SLA-based brokering service for the federated Cloud. However, their scheme is relevant primarily to the initial phase of the scheme presented in this chapter where the SLAs are defined, collected and ranked according to the customers' requirements. A taxonomy of the existing Inter-Cloud architectures and their brokering features is provided in [GB14]. The EU-funded OPTIMIS project [FHT+12] developed a toolkit based on agent technology to optimize the full service life cycle in the Cloud. A significant drawback of the OPTIMIS architecture, in terms of interoperability, is that the participating CSPs need to develop and maintain multiple vendor specific OPITMIS adapters to benefit from the entire toolkit. Another EU-funded project is mOSAIC [MOS16], which aims to simplify the development and deployment of Multi-Cloud applications. One of the shortcomings of this approach is that the developers need to rebuild their applications to integrate the mOSAIC API before using the framework.

2.1.2 Service Dependencies

Explicit knowledge about the dependencies is needed to support the management of SLAs by customers and CSPs. As proposed in W2.3 and [TMT+16], a service dependency is a directed relation between the services offered in the Cloud scenarios. It is expressed as a $1:n$ relationship

[5] In this chapter we do not address the SLA interoperability issue in a Multi-Cloud and refer the interested reader to [Far14] for further details.

[6] SLOs constitute the measurable elements of an SLA that specify the Cloud service levels required by the customers and to be achieved by the CSP.

where one service (termed as "dependent") depends on one or multiple services (termed as "antecedents"). A service can depend on data or resources provided by another service. A service s_1 is dependent on service s_2 if the provisioning of s_1 is conditional to the provisioning of s_2. We classify dependencies based on their occurrence between services at the same hierarchical level (horizontal dependencies), as well as between different levels in the hierarchy structure (vertical dependencies) [WS09]. Dependencies can be further classified into direct and indirect dependencies. Indirect dependencies occur between services which do not directly interact with each other, but where a transitive relationship exists via an intermediate service. In many cases horizontal and vertical dependencies occur at the same time and both dependencies affect the whole composition hierarchy.
We assume that the dependencies between services and SLOs in the SLA are predefined, and described using the suggested formats/guidelines from the relevant standards working groups. In these groups, usually industry and academia design the SLA contents and define the type of dependencies. These sets of dependencies are categorized in the SLA template. This template is used for the creation of the dependency model and for the SLO validation.
Most existing approaches to manage dependencies within SLAs focus on the handling of individual SLAs [LDK04, BPSMS07, AF08]. They do not consider information about dependencies between services in compositions. In this chapter, we present a dependency representation model which captures the information about services (each composed of a set of SLOs) and the dependencies that occur between them. This model is then used for validating composite services by checking the existence of service conflicts and SLO violations that occur due to different dependency relations between services. The process of analyzing conflicts is both interactive and iterative.

2.1.3 SLAs and the Multi-Cloud Environment

The varied stakeholders in the Cloud community have identified that specifying the measurable SLOs in SLAs is useful to provide and manage the assurance from two perspectives, namely (a) the service level being offered by a CSP, and (b) the service level requested by a CSC. Although the state of the art predominantly focuses on the methodologies to build and represent these SLAs [BPSMS07, AF08, GVB13, LLS12, TTLS14], most of these methodologies (a) focus on single Cloud infrastructure and on the best matching CSP selection and (b) do not take into account the information about dependencies between services. Overall, it is important to provide customers with a comprehensive support that can enable an automatic detection of conflicts and explanations for the dependent relations. Therefore, it is becoming an important issue for customers to make decisions regarding how to (a) assess CSP's qualitative and quantitative services, and (b) analyze the composite service dependencies to handle SLOs' violations and conflicts.

The Multi-Cloud perspective is considered in ESCUDO-CLOUD in the Use Cases 3 and 4. Storing information in a flexible way among different Clouds allows to increase performance and can also increase security by, for example, applying encryption techniques that split the information between the Clouds. However, there are some concerns that have to be considered when using different Cloud providers under the same Cloud service:

- **Compatibility**. We need to know if the interacting Clouds indeed have the compatibility to actually inter-operate. This is assumed beforehand for Use Case 4, where a federated approach is considered, and where the collaboration agreements between Clouds are assumed

to exist. This might not happen in Use Case 3, and thus the compatibility between Cloud providers becomes a critical issue.

- **Customer requirements**. Choosing which Cloud providers are more "convenient" for a Multi-Cloud storage service does not guarantee compatibility between Cloud providers. The customer requirements need to be met across all the providers. As a result we need mechanisms to assess the fulfillment of those requirements, being also able to decide what are the best providers to use in the Multi-Cloud composition.
- **Dependencies**. Dependency relates to both compatibility and customers requirements. While different Clouds might implement similar service metrics it might happen that there are dependencies between them (i.e., different encryption algorithms or different compliance of data protection regulations). This leads to potentially incompatible combinations of Cloud providers and might entail the violation of customers' requirements.

Over the recent years, the Multi-Cloud topics have become very active in the research community including ongoing EC projects. For example, MUSA[7] uses security metrics to check the fulfillment of the SLA and adapts the composition according to the monitoring events related to security metrics. However, in MUSA there is still no way to (i) evaluate the level of security provided by every provider of the Multi-Cloud, (ii) evaluate the dependencies between the components of a Multi-Cloud that might end up in conflicts, and (iii) choose the optimal composition based on a security assessment. Another Multi-Cloud project is Cumulus.[8] Cumulus works with a concept termed *certifications*, that are combined to create the SLAs of the Multi-Cloud. No security assessment is considered to manage any aspect of the Multi-Cloud service provision. Cumulus also defines a machine-readable SLA language, although the definition of such an SLA language is out of scope for ESCUDO-CLOUD. An interesting concept inspired by Cumulus is the consideration of vertical and horizontal composition that has been used in ESCUDO-CLOUD to analyze the dependencies of metrics and assess the providers of a Multi-Cloud to create the optimal SLA that fulfills the customer's requirements. Supercloud[9] also targets the topic of Multi-Cloud but from a functional point of view, addressing the definition of interfaces to allow the inter-operation of several Clouds. Although security is considered, as a constraint, in Supercloud to define the optimal performance of the system, the security controls used are very limited and focused on the optimal orchestration of the different Cloud providers. Finally, Cyclone[10] also addresses Multi-Cloud services but is primarily focused on techniques for the authentication between Clouds based on identity-management infrastructures.

2.2 ESCUDO-CLOUD Innovation

On this background, we develop an innovative scheme for the Multi-Cloud model that aims to solve the aforementioned issues by:

1. Offering a novel SLA-based Multi-Cloud service allocation approach that includes two main outcomes: (a) the Multi-Cloud SLA (i.e., MCSLA) construction, and (b) service selection and composition.

[7] http://www.tut.fi/musa-project/
[8] http://cumulus-project.eu/
[9] https://supercloud-project.eu/
[10] http://www.cyclone-project.eu/

2. Presenting a novel dependency representation model which captures the information about services and the dependencies that occur between them. This model is based on our initial work in W2.3 and [TMT+16] for validating CSP SLAs. This will be developed further in the upcoming deliverable D2.4. We enhance this model to fit into the Multi-Cloud environment and is used for validating the MCSLA by checking the existence of conflicts and SLO-level violations that occur due to different dependency relations across the Cloud services.

3. Validation of the proposed framework by evaluating actual CSP SLAs found on the public CSA STAR (Security, Trust and Assurance Registry) [The16b] repository.

Part of the work presented in this chapter has been published in [TMT+16].

2.3 The Proposed Framework

In this section we present a quantitative service-level assessment of CSPs to find the service composition that satisfies all the customer requirements. We assume that all transactions between the proposed framework and the CSPs are done through the Multi-Cloud middleware that communicates with the APIs of all involved CSPs. In our framework, the selection of composite services and the dependency management for composing the services is performed in five main stages as shown in Figure 2.2. After the CSPs submit their SLAs and the customers specify their requirements in Stage (A), the services assessment and selection algorithms are used to score services in Stage (B) according to the customer requirements. Based on this assessment, an optimum combination of services from multiple CSPs is chosen. This combination is used to create the MCSLA 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).
Before detailing the proposed framework stages, we note that customers are only able to trust the result of the proposed assessment as long as the information taken as input is reliable. In order to guarantee the validity of this model, the SLAs provided by the participating CSPs are required to come from a trusted source of information. In a real-world setup, the trust relationships can be given by an *Auditor* performing a third-party attestation of the CSP SLA (e.g., through a scheme such as the CSA Open Certification Framework (OCF) [The16a]). The audited SLAs are then stored by the CSPs in a trusted repository of SLAs (e.g., the CSA STAR repository [The16b]) as shown in Figure 2.2. This trust assumption relies on the fact that the certifications and the repository are trusted, and that the published SLAs were valid at the time of issuing the corresponding certification or publishing the information in the repository.

2.3.1 Stage A: Customer Requirements Definition

During this stage, the customers create their set of requirements and specify their preferences based on the same SLA template used by the CSPs to specify their offered services. Customers specify requirements at the SLO level by defining the required value for each SLO. The process of modeling values to a quantitative metric is not straightforward as SLOs can have varied types/ranges of qualitative and quantitative values. Hence, we use the notion of "service level" associated to each SLO of the SLA[11]. An SLA consists of a set of services $S = \{s_1, \ldots, s_n\}$. Each service consists

[11] We introduced the notion of "service level" in [TMT+16].

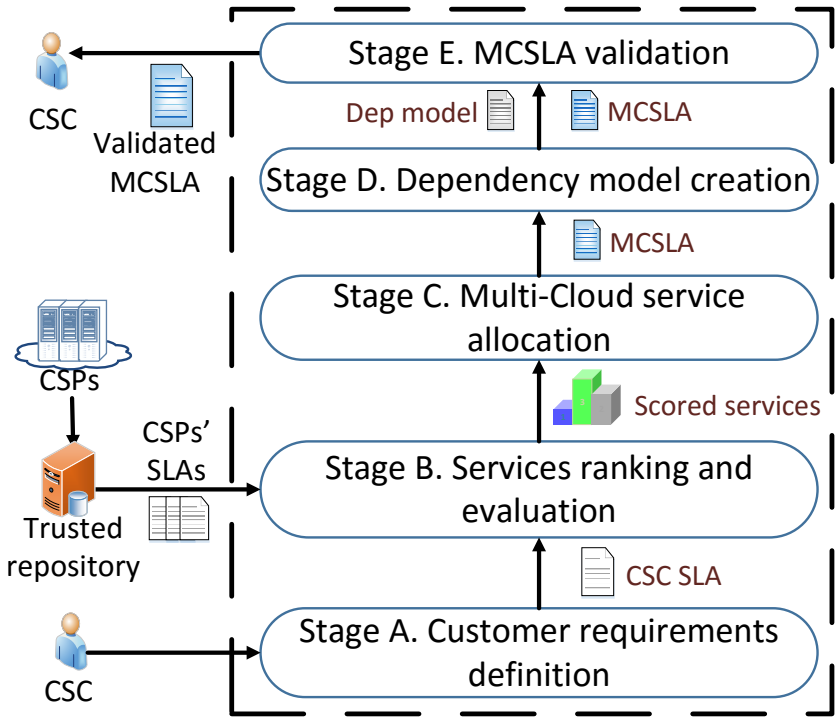


Figure 2.2: Proposed framework stages

of finite positive number o of SLOs k_i; where $i = 1,\ldots,o$. Each SLO k_i consists of m different metric values v_i; such that $v_{i,1}, v_{i,2}, \ldots, v_{i,m}$. Each value implies a different service level offered by the CSP and required by the customer. The total order of service levels of each k_i is defined using an order relation "$\prec_i$", such that $v_{i,1} \prec v_{i,2} \prec \ldots \prec v_{i,m}$. Each k_i value is mapped to a progressive numerical value according to its order. These numerical values are then normalized with respect to the k_i's number of metric values (m) $\frac{1}{m} \prec \frac{2}{m} \prec \ldots \prec \frac{m}{m}$.

2.3.2 Stage B: Services Evaluation

The quantitative service level assessment of CSPs (for their match to the customer requirements) is developed in this stage. Using this assessment, the services are scored based on the customer requirement level for each service. The SLA services assessment is performed in the following progressive phases:

Phase 1. Hierarchical Structure

The SLAs are modeled as a hierarchical structure as depicted in Figure 2.3, such that the first layer of the hierarchy is the Root level, which determines the overall rank. The second and third levels represent the main link to the services framework used by the CSP. The lowest level in the SLA structure represents the actual SLOs committed by the CSP, where threshold values are specified in terms of service metrics.

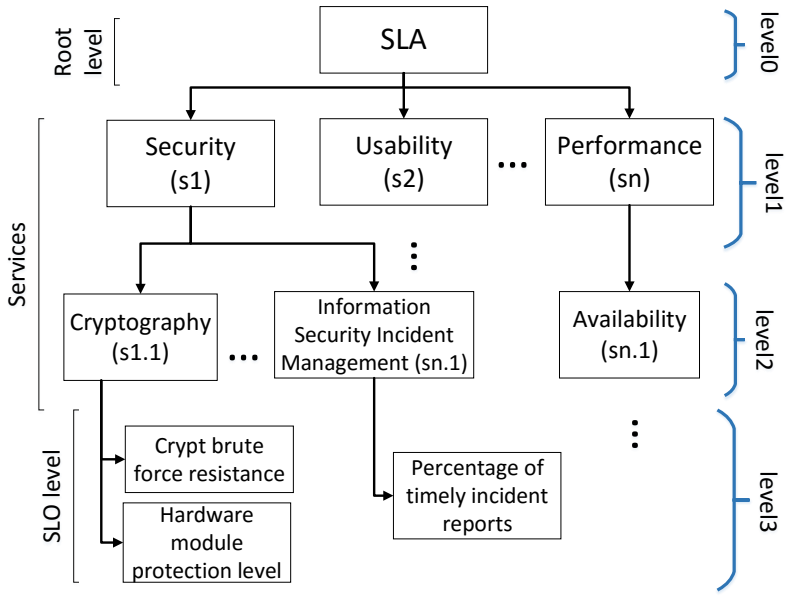


Figure 2.3: SLA hierarchy

Phase 2. Weights Assignment

In order to compare two CSP's SLOs, the relative importance levels of the customer requirements for each SLO are assigned as weights. We utilize qualitative terms to specify the importance of each SLO, where customers can assign qualitative terms as weights to each SLO to indicate their priorities (*Extremely-Important(EI)*, *Highly-Important(HI)*, *Low-Important(LI)*, and *Not-Required(NR)*). These qualitative terms are further transformed to quantitative values. The weights assignment is explained using a real-world case study in Section 2.4.

Phase 3. Services Quantification

In order to assess CSP's SLAs, a measurement model for different SLOs should be defined. We use a relative ranking model based on the Analytic Hierarchy Process (AHP) [Saa90]. The ranking model defines the most important requirements used and their quantitative values. AHP is a widely used method for solving problems related to Multi Criteria Decision Making (MCDM) [Zel82] such as comparing, ranking, and selecting multiple alternatives, when each alternative has multiple attributes. The advantages of AHP over contemporary multi-criteria methods are its ability to handle composite qualitative and quantitative attributes, along with its flexibility and ability to identify inconsistencies across requirements [Ram01]. The ranking model is based on a pairwise relation of services provided by CSPs and required by customers such that:

$$CSP_{1,k}/CSP_{2,k} = \frac{v_{1,k}}{v_{2,k}} \tag{2.1}$$

Each CSP_i offers SLO k with value v_i. Hence, $CSP_{1,k}/CSP_{2,k}$ indicates the relative rank of CSP_1 over CSP_2 for a particular SLO k. Similarly, $CSP_{1,k}/CSC_k$ indicates the relative rank of CSP_1 over the Cloud Service Customer CSC for k, which specifies whether CSP_1 satisfies CSC requirement or not. Thus, for each SLO we have a one-to-one Comparison Matrix (CM) of size $(n+1) \times (n+1)$

if there is a total of n CSPs and one CSC, so that:

$$CM_k = \begin{array}{c} \begin{array}{cccc} CSP_{1,k} & \dots & CSP_{n,k} & CSC_k \end{array} \\ \begin{pmatrix} CSP_{1,k}/CSP_{1,k} & \dots & CSP_{1,k}/CSP_{n,k} & CSP_{1,k}/CSC_k \\ \vdots & \ddots & \vdots & \vdots \\ CSP_{n,k}/CSP_{1,k} & \dots & CSP_{n,k}/CSP_{n,k} & CSP_{n,k}/CSC_k \\ CSC_k/CSP_{1,k} & \dots & CSC_k/CSP_{n,k} & CSC_k/CSC_k \end{pmatrix} \end{array} \tag{2.2}$$

Next the relative ranking of all the CSPs and the customer for each SLO is calculated as a priority vector[12] (PV_k) of the CM_k.

$$PV_k = \begin{array}{c} \begin{array}{ccccc} CSP_{1,k} & CSP_{2,k} & \dots & CSP_{n,k} & CSC_k \end{array} \\ \begin{pmatrix} N_{1,k} & N_{2,k} & \dots & N_{n,k} & N_{c,k} \end{pmatrix} \end{array} \tag{2.3}$$

$N_{1,k}$ is a numerical value representing the relative rank of CSP_1 to other CSPs as well as the CSC regarding an SLO k. $N_{c,k}$ is the relative rank of the CSC required service level with respect to the service levels offered by the CSPs. We demonstrate and validate the framework presented in this section using a real-world case study in Section 2.4.

Phase 4. Services Selection

Based on the relative ranking of CSPs according to the CSC requirements (determined in the previous phase), finding the best combination of SLOs that can collectively satisfy the customer needs is performed in this phase. The selection of the set of feasible services with respect to a set of customer requirements is performed using Algorithm 1. Each SLO priority vector (Equation 2.3) serves as an input to this algorithm to determine the services from CSPs that are compatible with the customer requirements.

Algorithm 1 Services selection for the case services are compatible

```
1: procedure MCSLASELECTION
2:     if N_{c,k} = N_{1,k} ∨ N_{2,k} ∨ ... N_{n,k} then                ▷ Condition 1
3:         N_{m,k} = N_{1,k} ∨ N_{2,k} ∨ ... N_{n,k}
4:     else if N_{c,k} < N_{1,k} ∨ N_{2,k} ∨ ... N_{n,k} then           ▷ Condition 2
5:         N_{m,k} = min(N_{1,k}, N_{2,k}, ... N_{n,k})
6:     else if N_{c,k} > N_{1,k} ∨ N_{2,k} ∨ ... N_{n,k} then           ▷ Condition 3
7:         N_{m,k} = max(N_{1,k}, N_{2,k}, ... N_{n,k})
8:     end if
9: end procedure
```

In the algorithm we describe different conditions for services selection in comparison with the customer requirements, which we explain using an example. Consider an SLO k offered by three providers with three different service levels values $v_{1,k}$, $v_{2,k}$, $v_{3,k}$ and required by a customer with value $v_{c,k}$. After, the PV_k is calculated using Equation 2.3 the following conditions take place:
First Condition: If $N_{c,k}$ is equal to any of the three relative ranking values (e.g., $N_{c,k} = N_{3,k}$), then the MCSLA numerical relative ranking value ($N_{m,k}$) will be assigned the same value as $N_{3,k}$ which

[12]The priority vector is an approximation eigenvector of the comparison matrix which indicates a numerical ranking of providers that specifies an order of preference among them, as indicated by the ratios of numerical values.

means CSP_3 is selected to provide the required service.
Second Condition: If the first condition is not satisfied and $N_{c,k}$ is lower than the CSPs (e.g. $N_{c,k} < N_{1,k}$ and $N_{2,k}$, which means $CSP_{1,k}$ and $CSP_{2,k}$ are over-provisioning the customer's requirement). Then $N_{m,k}$ is equal to the minimum of the over-provisioning CSPs values (N_m is equal to the closest value to the customer required value).
Third Condition: If neither the first nor the second conditions are satisfied, then the providers are under-provisioning the customer requirement. Thus, the provider with the highest offered level is chosen (the one closest to the customer's requirement).

Phase 5. Services Aggregation

In this phase, we follow up with a bottom-up aggregation to give an overall assessment of the service levels. To achieve that, the priority vector of each SLO (Phase 3) is aggregated with their relative normalized weights assigned in Phase 2. This aggregation process is repeated for all the SLOs in the hierarchy with their relative weights, which results in the ranking of all the Cloud providers based on customer-defined requirements and weights.

$$PV_{aggregated} = \begin{bmatrix} PV_{k_1} & \dots & PV_{k_n} \end{bmatrix} . \begin{bmatrix} W \end{bmatrix}^T \tag{2.4}$$

W is the set of normalized weights of different SLOs such that $W = w_{k_1}, w_{k_2}, \dots, w_{k_n}$. Note that the weights are normalized to satisfy the AHP requirements [Saa90]. PV_{k_1} is the PV calculated for SLO k_1.

2.3.3 Stage C: Multi-Cloud Service Allocation

After finding the best combination of services that collectively satisfy "all" the customer needs in Stage B, mapping this combination to the Multi-Cloud SLA (named MCSLA) is the target of this stage. This is done by constructing the MCSLA template and then using the services scores provided in the previous stage to build the composition of the feasible services. The main purpose of constructing such an MCSLA is addressing the SLA interoperability issue in a Multi-Cloud.
In order to automatically and quantitatively evaluate the MCSLA, we use the classical "AND-OR" trees to identify the set of services that satisfy the customer requirements as depicted in Figure 2.4. This is done by forming compositions of services represented by each "AND" or "OR". As inferred from their name, "AND" relationships represent the "necessary" customer requirements and "OR" relationships are more adequate to model "optional" requirements. Furthermore, if more than one provider satisfies the customer requirement for a specific service, these providers are mapped using "OR" relationships and then based on other criteria (e.g., cost, previous customers rating, etc.) the best CSP is selected (i.e., the CSP with the lowest cost for example) as depicted in Figure 2.4.
After mapping the selected services to the MCSLA, a dependency model is created in the next stage to capture information about MCSLA services and the dependencies that occur between them.

2.3.4 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. In order to model services dependencies, it is important to (i) specify on

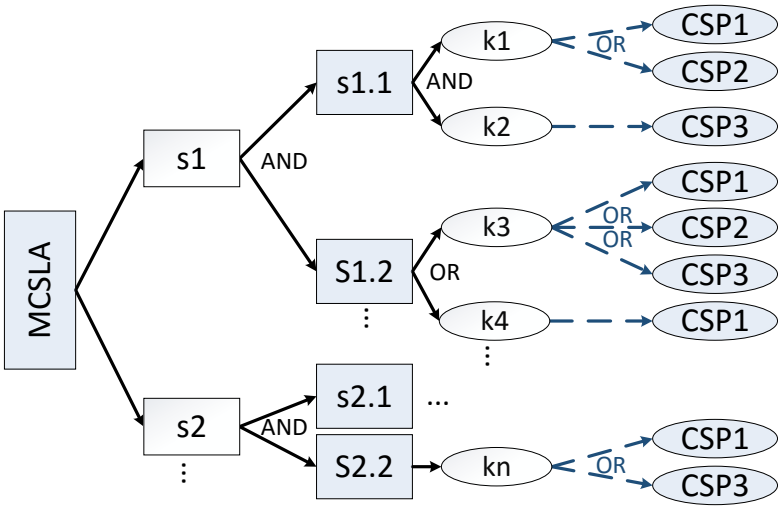


Figure 2.4: MCSLA "AND-OR" tree

which SLOs the fulfillment of a specific service depends and how it depends on these SLOs, (ii) derive the expected requirements the dependency model should support, such as:

1. *Support of different dependency types*. The dependency model should support different types of dependencies as well as various dependencies classifications (e.g., unidirectional and bidirectional dependencies).

2. *Support of multiple dependencies*. One service can depend on several other services. These dependencies could be the same or of different types.

3. *Dependency model validation*. It should be possible to automatically validate the dependency model to avoid inconsistencies and conflicts.

We model a *MCSLA* by a tuple $MCSLA = (S, l, \rightarrow_S, K, \rightarrow_K, v)$, such that:

- S is a set of services s (i.e., composition of services offered by multiple CSPs as per their SLAs) such that $S = s_{i,1} \odot s_{i,2} \odot \ldots \odot s_{i,m}$; $i = 1 \ldots n$ where n is the number of CSPs and m is the number of services.
  - $s_{i,j}$ specifies service s_j provided by CSP_i
  - $\odot$ is a generic composition operator, e.g., $s_1 \odot s_2$ denotes the composite service formed by composing services s_1 and s_2 in some way. We are interested in which services are being composed but not how they are being composed, thus we model the composition process using a generic composition operator that can represent any type of composition.

- *MCSLA* is associated with hierarchical levels $l(s) \in \{0, 1, \ldots, n-1\}$. It contains exactly one service s with $l(s) = 0$, which is the root service. K is a set of SLOs with associated hierarchy level $l(k) = n$.

- $\rightarrow_S \subseteq S \times (S \cup K)$ models service dependencies.

- For two CSPs, CSP_i and CSP_j, we write $s_{i,1} \rightarrow_S s_{j,2}$ if $s_{i,1}$ (dependent service) depends on $s_{j,2}$ (antecedent service) where $i \in \{1 \ldots n\}$ and $j \in \{1 \ldots n\}$. If $i = j$ then s_1 and s_2 are provided by the same CSP.

- Each service is decomposed into a Boolean combination of SLOs $k_1 \otimes k_2 \otimes \ldots \otimes k_o$; where $\otimes \in \vee, \wedge$ and o is the number of SLOs.

- K is a set of SLOs with associated hierarchy level $l(k) = n$ for all $k \in K$ where:

  - Each SLO has a unique ID (i.e., SLO_{id}).
  - Each SLO has an SLA ID (i.e., SLA_{id}). The SLA ID specifies the CSP providing this SLO.
  - $\rightarrow_K \subseteq K \times K$ models SLO dependencies. We have $k_{i,1} \rightarrow_K k_{j,2}$ if $k_{i,1}$ (dependent SLO) depends on $k_{j,2}$ (antecedent SLO).
  - $v : K \mapsto V$ is an assignment of values in V to SLOs, where V is the set of all metric values of each SLO in K.
  - $\forall k \in K \mid \nexists k \in K : v_{j,k} \succ v_{i,k}$ models the ordering over the possible CSPs for each SLO using a preference relation $\succ$. $v_{i,k}$ and $v_{j,k}$ represent CSP_i and CSP_j offered values for an SLO (k). We define $v_{i,k} \succ v_{j,k}$ if only CSP_i satisfies the customer requirement. If both CSPs satisfy the customer requirement regarding SLO k ($v_{i,k} = v_{j,k}$) then the one with min cost is chosen.
  - A constraint on SLO dependency relation is specified using a constraint set $C_v^{\rightarrow_K} \subseteq K \times K \times \{=, \neq, <, \leq, >, \geq\}$. A constraint $(k_{i,1}, k_{j,2}, \equiv) \in C_v^{\rightarrow_K}$ is satisfied if the values of $k_{i,1}$ and $k_{j,2}$ are related by the given comparison, i.e., $v(k_{i,1}) \equiv v(k_{j,2})$. A dependency relation $k_{1,i} \rightarrow_K k_{2,j}$ is called valid, written $valid_{C_v}{\rightarrow_K}(k_{i,1}, k_{j,2})$, if the relation satisfies all its constraints, i.e., $\forall (k'_{i,1}, k'_{j,2}, \equiv) \in C_v^{\rightarrow_K}.(k_{i,1} = k'_{i,1}$ and $k_{j,2} = k'_{j,2}) \Rightarrow v(k_{i,1}) \equiv v(k_{j,2})$.

- We write the transitive closure of $\rightarrow_S$ as $\rightarrow_S^+$, i.e., $s_{i,1} \rightarrow_S^+ s_{j,2}$ if $\exists s_{i,3} \in S.s_{i,1} \rightarrow_S s_{i,3}$ and $s_{i,3} \rightarrow_S^+ s_{j,2}$. A dependency $f_{i,1} \rightarrow f_{j,2}$, where f is either a service or an SLO and $\rightarrow \in \{\rightarrow_S, \rightarrow_K\}$, between two objects $f_{i,1}, f_{j,2} \in S \cup K$ is called symmetric if also $f_{j,2} \rightarrow f_{i,1}$. Otherwise, $f_{i,1} \rightarrow f_{j,2}$ is called non-symmetric. For example, f_1 and f_2 are symmetrically dependent if $f_{i,1} \rightarrow f_{j,2}$ and $f_{j,2} \rightarrow f_{i,1}$.

- A *MCSLA* $(S, l, \rightarrow_S, K, \rightarrow_K, v)$ has to satisfy the following constraints:

  i) Services do only depend on services of the same or the next lower hierarchy level: $\forall s_{i,1}, s_{j,2} \in S.s_{i,1} \rightarrow_S s_{j,2} \Rightarrow l(s_{i,1}) = l(s_{j,2})$ or $l(s_{i,1}) + 1 = l(s_{j,2})$

  ii) Services do not depend on themselves: $\forall s \in S.\neg s \rightarrow_S s$

  iii) All services depend directly or indirectly on an SLO: $\forall s \in S \exists k \in K.s \rightarrow_S^+ k$

A meta-model is then developed based on the presented dependency definitions. This meta-model allows the description of services along with information on the MCSLA drafted for it.[13] The meta-model is specified using a machine readable format (allowing fully automatic validation) such as an XML data structure using an XML Schema. In this Schema, service dependency

[13] We assume that all dependencies between services and SLOs in the SLA are predefined and described using relevant scientific and multidisciplinary working groups as specified in Section 2.1.2.

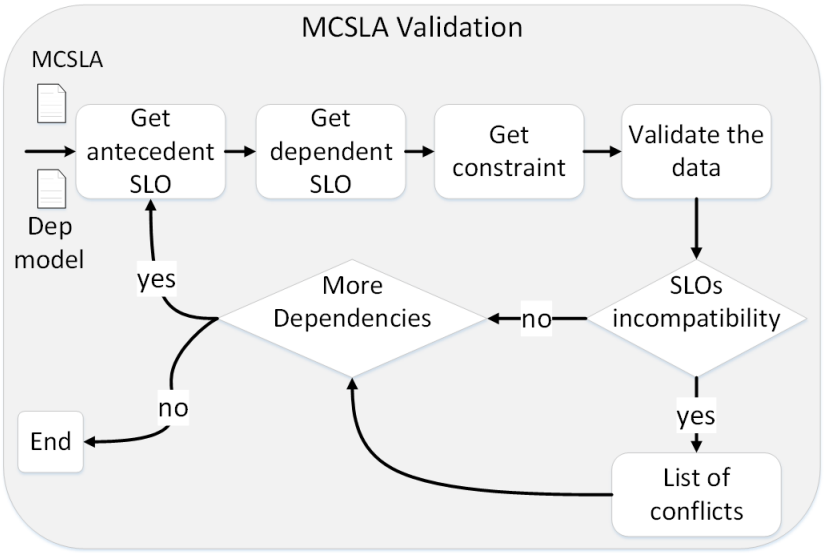


Figure 2.5: Dependency based MCSLA validation stages

information is modeled including the involved SLOs and their roles as dependent or antecedent as well as the SLOs values.
Following the MCSLA and dependency model construction, the MCSLA is validated to check service conflicts and different SLOs compatibility issues.

2.3.5 Stage E: MCSLA Validation

As depicted in Figure 2.5, the validation is done by first extracting the dependency model ID, SLO ID, SLA ID, and the dependency ID of each two dependent SLOs[14] defined in the XML schema. Furthermore for each dependency relation, the antecedent and dependent SLO values are extracted. This entails extracting the constraint comparative (i.e., $=, \neq, <, \leq, >, \geq$) and checking if the two dependent SLOs' values satisfy the constraint. If the constraint between dependent SLOs is not satisfied, the validation scheme shows conflict between these two SLOs. The SLO ID, SLA ID, dependency ID and dependent SLO ID of the affected SLO are saved in a list, while the evaluation is continued to determine further conflicts. At the end of this process 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. Algorithm 2 details the necessary steps for extracting the values of validating a dependency relation between SLOs in pseudocode notation.

2.4 Case Study: Evaluation of CSP's SLAs based on STAR repository values

This section shows an empirical validation of the proposed framework through two scenarios that use real world SLA information derived from the Cloud Security Alliance's STAR repository [The16b]. This initial validation scenario demonstrates how a Cloud customer can apply the framework presented in this section to compare side-by-side three different CSPs based on their

[14]Each SLO has a unique ID and an SLA ID. Moreover, each dependency relation in the same dependency model has a unique ID.

Algorithm 2 Validation

```
Input: String mcslaID, String depmodelID, String depID, String depSLO, String antSLO,
String depConst
Output: List affectedSLOs
List depList = NIL
Value depValue = NIL
Value antValue = NIL
List depModels = getModels(depmodelID, mcslaID)
for model ∈ depModels do
    depList = getDependencies(depID, depmodelID)
    for depend ∈ depList do
        depValue=getValue(depend.depSLOvalue, depID)
        antValue=getValue(depend.antSLOvalue, depID)
        depConst=getString(depend.constraint, depID)
        if validate(depValue, antValue, depConst) ≠ true then
            affectedSLOs.add(depend.depSLO)
        end if
    end for
end for
```

advertised SLAs (compliant with the hierarchy of Figure 2.3). Table 2.1 presents a sample dataset used for this scenario, based on the information available in the CSA STAR repository, where the values associated to 15 SLO metrics for the three selected CSPs are presented. In order to perform a comprehensive validation, the selected SLOs comprised both qualitative and quantitative metrics. The Qualitative metrics are specified as service levels using Definition 1 such as $monthly, weekly$, and $daily$ denoted as service levels $level_1$, $level_2$ and $level_3$, which are modeled as $\frac{1}{3}, \frac{2}{3}, \frac{3}{3}$. Furthermore, no, yes metrics are denoted as $level_0$, $level_1$ respectively. All CSP SLOs are normalized to the customer requirements to eliminate masquerading.[15] Weights assigned by the customers to indicate their priorities are specified as numerical value such that *EI* and *NR* indicate a relative value *1* and *0* respectively. *HI* and *LI* can be considered any intermediate values between *1* and *0*. In this analysis they indicate a relative rank value *0.7* and *0.3* respectively.

[15]The masquerading effect happens when the overall aggregated security level values mostly depend on those security controls with a high-number of SLOs, thus negatively affecting groups with fewer, although possibly more critical, provisions.

Cloud SLA based on CSA STAR [The16b]					CSPs			Customer (CSC)	
Services			SLOs		CSP_1	CSP_2	CSP_3	req	weight
			name	dep.					
Root	Audit & Compliance *AC*	Planning *AC*1	*AC*1.1	Dep_1	*yes*	*no*	*yes*	*yes*	EI
			*AC*1.2		$level_2$	$level_3$	$level_2$	$level_3$	HI
		Independent Audits *AC*2	*AC*2.1	Dep_1	*yes*	*yes*	*no*	*yes*	HI
				Dep_2					
			*AC*2.2	Dep_3	*no*	*yes*	*yes*	*no*	NR
			*AC*2.3	Dep_3	*no*	*yes*	*yes*	*yes*	EI
				Dep_4					
			*AC*2.4		*yes*	*no*	*yes*	*yes*	EI
			*AC*3.1		*no*	*yes*	*no*	*yes*	EI
			*AC*3.2		*no*	*yes*	*yes*	*yes*	EI
		Regulatory Mapping *AC*3	*AC*3.1	Dep_2	*no*	*yes*	*no*	*yes*	EI
			*AC*3.2	Dep_4	*no*	*yes*	*yes*	*yes*	EI
			*AC*3.3		$level_2$	$level_1$	$level_3$	$level_3$	LI
	Business Continuity *BC*	Testing *BC*2	*BC*2.1		*yes*	*yes*	*yes*	*no*	NR
			*BC*2.2		*no*	*yes*	*no*	*no*	NR
		Policy *BC*11	*BC*11.1		*yes*	*yes*	*yes*	*yes*	LI
			*BC*11.2		$level_3$	$level_2$	$level_3$	$level_3$	LI
		Regulatory Mapping *AC*3	AC3.1		*no*	*yes*	*no*	*yes*	EI
			*AC*3.2		*no*	*yes*	*yes*	*yes*	EI
			AC3.3		$level_2$	$level_1$	$level_3$	$level_3$	LI
	Interface Security *IS*	Application Security *IS*1	*IS*1.1	Dep_5	*weekly*	*weekly*	*weekly*	*weekly*	EI
			*IS*1.2	Dep_5	$level_1$	$level_2$	$level_1$	$level_2$	EI

Table 2.1: Excerpt of SLA's from CSPs and customer requirements.

We consider dependencies between services and SLOs which are going to be validated using the Multi-Cloud validation model presented in Section 2.3.4 such that:
SLOs dependencies:

- $AC2.1$ depends on $AC1.1$ (this dependency relation is named as Dep_1 in Table 2.1). This is modeled as $AC2.1 \rightarrow_K AC1.1$ with constraint $(AC2.1, AC1.1, =) \in C_v^{\rightarrow_K}$.
- In the same way, Dep_2, Dep_3 and Dep_4 are specified as $AC2.1 \rightarrow_K AC3.1$ with $(AC2.1, AC3.1, =) \in C_v^{\rightarrow_K}$), $AC2.2 \rightarrow_K AC2.3$ with $(AC2.2, AC2.3, =) \in C_v^{\rightarrow_K}$ and $AC2.3 \rightarrow_K AC3.2$ with $(AC2.3, AC3.2, =) \in C_v^{\rightarrow_K}$ respectively.
- Finally, $IS1.1$ and $IS1.2$ are symmetrically dependent (i.e., Dep_5) such that $IS1.1 \rightarrow_K IS1.2 \wedge IS1.2 \rightarrow_K IS1.1$ with constraint $(IS1.1, IS1.2, =) \in C_v^{\rightarrow_K}$.

Service dependencies:

- AC depends on $AC1$, $AC2$ and $AC3$. Such that $AC \rightarrow_S AC1 \wedge AC \rightarrow_S AC2 \wedge AC \rightarrow_S AC3$.
- BC depends on $BC2$, $BC11$ and $AC3$. Furthermore, IS depends on $IS1$. Each is further depending on the SLOs as shown in Table 2.1.
- $AC2$ depends on SLOs $AC3.1$ and $AC3.2$ (shaded in Table 2.1).
- Since BC depends on $AC3$, and $AC3$ depends on SLOs $AC3.1$, $AC3.2$ and $AC3.3$ then using transitive closure $BC \rightarrow_S^+ AC3.1 \wedge BC \rightarrow_S^+ AC3.2 \wedge BC \rightarrow_S^+ AC3.3$ (shaded in Table 2.1).

The assessment and evaluation process for the Cloud SLOs defined in Table 2.1 is explained step-by-step, in the rest of this section. For this evaluation technique, the customer specifies his/her requirements at the lowest level of the SLA (i.e., SLO level) and considers different relative importance (i.e., weights) for all of the SLOs. For the *Audit & Compliance* control of Cloud SLA, there are three sub-services ($AC1, AC2$ and $AC3$) which are further divided to SLOs ($AC1.1$, $AC1.2$, $AC2.1$, ...). For $AC1.2$ the providers and the customer can specify their SLOs values from $level_1$ to $level_3$. Using the data shown in Table 2.1, Equation 2.1 is used to define the $AC1.2$ pairwise relation such that:

$$CSP_{1,AC1.2}/CSP_{2,AC1.2} = \frac{2}{3}/\frac{3}{3}, \;\; CSC_{AC1.2}/CSP_{3,AC1.2} = \frac{3}{3}/\frac{2}{3}$$

Thus, the CM of $AC1.2$ is calculated using Equation 2.2 as:

$$CM_{AC1.2} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 1 & 2/3 & 1 & 2/3 \\ 3/2 & 1 & 3/2 & 1 \\ 1 & 2/3 & 1 & 2/3 \\ 3/2 & 1 & 3/2 & 1 \end{pmatrix} \end{array}$$

The relative ranking of the CSPs for $AC1.2$ is given by the priority vector of $CM_{AC1.2}$ ($PV_{AC1.2}$) which is calculated using Equation 2.3.

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} N_1 & N_2 & N_3 & N_c \end{array} \\ \begin{pmatrix} 0.2 & 0.3 & 0.2 & 0.3 \end{pmatrix} \end{array}$$

This implies that only CSP_2 satisfies the customer requirement for $AC1.2$ as $N_{2-AC1.2} = N_{c-AC1.2}$. Using Algorithm 1, since $N_{c-AC1.2}$ is equal to $N_{2-AC1.2}$ then $N_{m-AC1.2}$ is equal to $N_{2-AC1.2}$, this means CSP_2 is selected for providing *AC1.2* SLO according to the customer requirement.
Similarly, we calculate $CM_{AC1.1}$, $PV_{AC1.1}$ and $N_{m-AC1.1}$ such that:

$$PV_{AC1.1} = \begin{matrix} N_1 & N_2 & N_3 & N_c \end{matrix} \begin{pmatrix} 0.333 & 0 & 0.333 & 0.3333 \end{pmatrix}$$

This implies that both CSP_1 and CSP_3 satisfy the customer requirement for $AC1.1$. Using Algorithm 1, $N_{m-AC1.1}$ is equal to $N_{1-AC1.1}$ and $N_{3-AC1.1}$, this means either CSP_1 or CSP_3 is selected for providing *AC1.1* SLO according to the customer requirement. Thus the customer can choose any of two providers according to other factors such as cost or previous customers feedback.
The $AC1$ priority vector is then premeditated by aggregating $PV_{AC1.1}$ and $PV_{AC1.2}$ with the normalized weights from the customer specified *LI* and *HI* for *AC1.1* and *AC1.2* respectively as:

$$w_{AC1} = \begin{matrix} AC1.1 & AC1.2 \end{matrix} \begin{pmatrix} \frac{1}{1.7} & \frac{0.7}{1.7} \end{pmatrix}$$

PV_{AC1} is then calculated using Equation 2.4 such that:

$$PV_{AC1} = \begin{matrix} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{matrix} \begin{matrix} \begin{matrix} PV_{AC1.1} & PV_{AC1.2} \end{matrix} \\ \begin{pmatrix} 0.333 & 0.2 \\ 0 & 0.3 \\ 0.333 & 0.2 \\ 0.333 & 0.3 \end{pmatrix} \end{matrix} \begin{matrix} w_{AC1} \\ \begin{pmatrix} 0.588 \\ 0.412 \end{pmatrix} \end{matrix}$$

Therefore, this results into

$$PV_{AC1} = \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \begin{pmatrix} 0.278 & 0.124 & 0.278 & 0.32 \end{pmatrix}$$

This means that no provider is fully offering *AC1* services regarding the customer requirements. Both CSP_1 and CSP_3 are satisfying customer requirement *AC1.1* and only CSP_2 satisfies customer requirement *AC1.2*. Therefore, a composition of services from different providers satisfies the *AC1* customer requirements. The service composition for $AC1$, specified as $k_{AC1.1} \wedge k_{AC1.2}$, is (($CSP_1$[16] OR CSP_3) AND (CSP_2)) as shown in Figure 2.6.
In a similar way as in *AC1*, PV_{AC2} is calculated. Where *AC2* is composed of six SLOs $(k_{AC2.1} \wedge k_{AC2.2} \wedge k_{AC2.3} \wedge k_{AC2.4}) \wedge (k_{AC3.1} \vee k_{AC3.2})$ as shown in Figure 2.6 such that:

$$PV_{AC2} = \begin{matrix} PV_{AC2.1} & PV_{AC2.3} & PV_{AC2.4} & PV_{AC3.1} & PV_{AC3.2} \end{matrix} \begin{pmatrix} 0.333 & 0 & 0.3333 & 0 & 0 \\ 0.333 & 0.3333 & 0 & 0.5 & 0.3333 \\ 0 & 0.3333 & 0.3333 & 0 & 0.3333 \\ 0.333 & 0.3333 & 0.3333 & 0.5 & 0.3333 \end{pmatrix}$$

Thus, the best service combination for satisfying customer requirement *AC2* is offered by ((CSP_1 OR CSP_2) AND (CSP_2 OR CSP_3) AND (CSP_1 OR CSP_3) AND CSP_2). Note that *AC2.2* is not

[16] CSP_1 here means the service offered by CSP_1 which is *AC1.2*.

required by the customer so it is not offered in the Multi-Cloud composite service as shown in Figure 2.6.
Similarly, *Regulatory Mapping* priority vectors ($PV_{AC3.1}$, $PV_{AC3.2}$ and $PV_{AC3.3}$) are calculated and then $N_{m-AC3.1}$, $N_{m-AC3.2}$ and $N_{m-AC3.3}$ are determined. In a similar way the *Testing*, *Policy*, *Regulatory Mapping* and *Application Security* PVs are considered.
The set of SLOs that satisfy the customer requirements are selected as shown in the MCSLA "AND-OR" tree which shows the CSPs fulfilling the customer requirements for each SLO using AND/OR relations (i.e., Figure 2.6). Each SLO is specified as shown in Figure 2.6 in order to have the overall MC composition. After the set of SLOs selection the MCSLA is validated to detect SLOs conflicts.

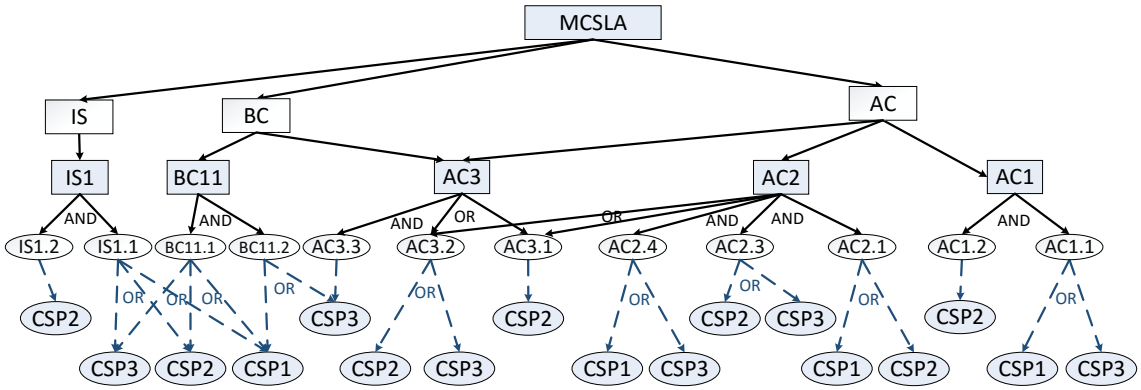


Figure 2.6: MCSLA regarding customer requirements

MCSLA Validation

Errors are automatically detected based on the modeled dependencies, such that:

- Dep_1 **Validation**: $CO2.1$ service level ($level_1$) is *equal* to the $CO1.1$ service level ($level_1$), $v(CO2.1) = v(CO1.1)$. **Result:** Valid for CSP_1 only. CSP_2 shows a conflict in Dep_1. Thus, only CSP_1 is selected for offering *CO2.1*.

- Dep_2 **Validation**: $AC3.1$ service level ($level_1$) is *equal* to $AC3.1$ service level ($level_1$). **Result:** Valid for CSP_2 however CSP_1 shows a conflict in Dep_2. Thus, only CSP_2 is selected for offering $AC3.1$. In the same way, Dep_3 and Dep_4 are validated.

- Dep_5 **Validation**: $IS1.1$ service level (*weekly* which is $level_2$) is *equal* to the $IS1.2$ service level ($level_2$). **Result:** Valid as CSP_2 the one chosen to offer $IS1.1$ and $IS1.2$ satisfy the constraint.

At the end of the validation process, a list of all conflicts found in the MCSLA with the conflicts explanation is created which allows these conflicts to be understood and resolved. Using MCSLA "AND-OR" tree, the MCSLA can simply remove the SLO that is causing conflict (offered by a specific CSP) and chooses another CSP to offer this SLO and validate the MCSLA again to detect any further conflicts.

2.5 Summary

The popularity of the Multi-Cloud model is on the rise given the larger variety, improved availability of services and also the new functionalities available for the customers, e.g., selecting services from multiple independent providers that best satisfy their requirements versus a single provider offering. However, despite the economic and performance advantages advocated for the Multi-Cloud, the actual process of selecting a set of suitable services from different providers is still a challenging task. Given the increasing number of CSPs that offer a variety of services with different capabilities and prices, the customers now need to (a) be able to parse the service specifications and also (b) comparatively asses this variety of offered services, to be able to choose the composition of services that best match their requirements. Such solutions for service selection and SLA management for the Multi-Cloud model are currently lacking. In this chapter, we have introduced a novel SLA-based service selection methodology for the Multi-Cloud environment that provides a MCSLA construction, and also a related services selection and management process. We have investigated the MCSLA approach while considering dependencies occurring across different SLOs. Furthermore, we introduced a ranking algorithm to score services offered by the CSPs, as based on their SLAs and according to the customer requirements. Our evaluation, based on actual case studies, shows that our approach effectively selects a set of services, for the overall service composition, that satisfies the customer's requirements.

3. Data Distribution and Swapping for Access Confidentiality

The Cloud is today characterized by the presence of a rich variety of Cloud providers offering storage and computational services. This can be seen as an opportunity for users to operate over multiple providers, thus enhancing security while maximizing efficiency of accesses.

In this chapter, we present a technique that takes advantage of multiple providers to protect data and access confidentiality. Building on the shuffle index approach studied in Work Package 2, we propose a technique that randomly partitions data among three independent Cloud providers and ensures access confidentiality by dynamically re-allocating data at every access. Dynamic re-allocation is enforced by swapping data involved in an access across the providers, in such a way that accessing a given node implies re-allocating it to a different provider. This protection technique destroys the ability of providers to build knowledge by observing accesses. The use of three Cloud providers ensures uncertainty of the result of the swapping operation, even in presence of collusion among them.

The remainder of this chapter is organized as follows. Section 3.1 presents the state of the art. Section 3.2 discusses the innovation provided by ESCUDO-CLOUD. Section 3.3 recalls the basic concepts of the shuffle index. Section 3.4 introduces the rationale of our approach. Section 3.5 describes our index structure working on three providers. Section 3.6 presents the working of our approach, discussing protection techniques and data access. Section 3.7 analyzes protection guarantees. Section 3.8 discusses the motivations behind our choice of swapping and of three as the number of providers to be used, and provides some economic considerations for our approach. Finally, Section 3.9 concludes the chapter.

3.1 State of the Art

The problem of protecting data confidentiality when moving to the Cloud is today widely recognized and has received considerable attention by the research and development communities in the last few years (e.g., [SD16, SD10, CDF+10, HQL+11]). The proposed approaches are based on encryption and fragmentation techniques, or on their combined adoption, and guarantee efficiency of access operations while protecting the sensitive content of the data [PFL15]. Specific indexing (e.g., [HIML02, SD10]) and keyword-based search techniques (e.g., [WCRL12]) have been developed to support efficient access to encrypted data.

As already pointed out in Deliverable D2.1, simply protecting the confidentiality of data content is not sufficient. Indeed, accesses to the data could reveal sensitive information, also about the data content itself. The protection of access and pattern confidentiality (i.e., the confidentiality of the specific accesses performed on the data and sequences thereof, respectively) is therefore equally important, and several approaches addressing this problem have been proposed [DFLS16, DFLS15, DFS15a, DFS15b, DFM+16, DFP+16]. Solutions for protecting access

and pattern confidentiality are based on Private Information Retrieval (PIR) techniques. Such solutions, however, do not protect content confidentiality and suffer from high computational costs (e.g., [OS07]), even when different copies of the data are stored at multiple non-communicating servers (e.g., [CMS99]). Recent approaches address the access and pattern confidentiality problems through the definition of techniques that dynamically change, at every access, the physical location of the data. Some proposals have investigated the adoption of the Oblivious RAM (ORAM) structure (e.g., [WSC08]), in particular with recent proposals aimed at making ORAM more practical such as ObliviStore [SS13b], Path ORAM [SvS+13], and Melbourne Shuffle [OGTU14]. ORAM has also been recently extended to operate in a distributed scenario [LO13, SS13a]. The goal of these solutions is to reduce communication costs for the client and then make ORAM-based approaches available also to clients using lightweight devices. The privacy guarantees provided by distributed ORAM approaches however rely on the fact that storage servers do not communicate or do not collude with each other. Alternative solutions are based on the adoption of a tree-based structure (e.g., [LC04, YZZQ11]) to preserve content and access confidentiality.

The shuffle index has been first introduced in [DFP+11] and extended in [DFP+13a, DFP+13b] to accommodate concurrent accesses on a shuffle index stored at one storage server. A preliminary investigation of the distribution of the shuffle index has been presented in [DFP+14]. We have built on this preliminary result, and on the results of Work Package 2, by: formalizing and providing a comprehensive analysis of the protection provided by the distributed shuffle index, and by analyzing and studying the choice of distributing the shuffle index among three providers.

3.2 ESCUDO-CLOUD Innovation

The technique discussed in this chapter distributes the data structure over three different Cloud providers. Each provider will store only a portion of the index structure and will then be able to observe only a portion of the accesses over it. Access and pattern confidentiality is then obtained by properly allocating data at initialization time, and by dynamically moving accessed data among the Cloud providers as a consequence of each search operation. The proposed solution has been designed considering the fact that Cloud providers may collude and exchange information among each other to gain access to sensitive data.

The distribution of the shuffle index among three providers and the adoption of swapping for protecting access and pattern confidentiality have been first proposed in [DFP+14]. The work of ESCUDO-CLOUD project has considerably extended our prior work in the following directions.

- We have analyzed the protection provided by the distributed shuffle index approach, to measure the quality of the proposed solution. To this purpose, we studied how our solution is able to guarantee access and pattern confidentiality, by measuring how quickly information is destroyed and how effective our approach prevents knowledge accumulation in a Multi-Cloud scenario. In our analysis, we considered two complementary scenarios, analyzing how quickly information possessed by providers degrades as a consequence of shuffling, and the limited information gain that providers can acquire by observing known accesses to the index structure.

- We have analyzed and studied the choice of having three providers, clearly motivating our choices in the definition of the proposed protection techniques. We also discussed the advantages obtained when distributing the shuffle index among three providers.

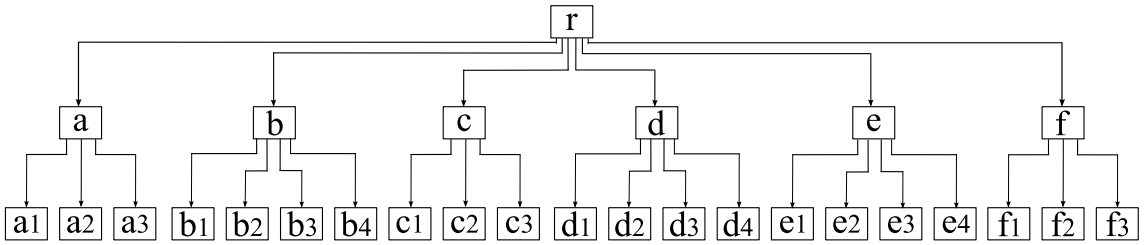


Figure 3.1: An example of unchained $B+$-tree

- We have analyzed the economic advantages of our solution, which is based on the adoption of multiple Cloud providers.

The work presented in this chapter has been published in [DFP+15b].

3.3 Basic Concepts

The *shuffle index* [DFP+15a, PFL15] is a dynamically allocated data structure offering access and pattern confidentiality while supporting efficient key-based data organization and retrieval. At the *abstract* level, a shuffle index is an *unchained B+-tree* (i.e., there are no links between the leaves) with fan-out F. Since the shuffle index is a $B+$-tree: *i)* each node stores up to $F-1$ ordered values and has as many children as the number of values stored plus one; *ii)* the tree rooted at the j-th child of an internal node stores values included in the range $[v_{j-1}, v_j)$, where v_{j-1} and v_j are the $(j-1)$-th and j-th values in the node, respectively; and *iii)* all leaves, which store actual tuples, are at the same level of the tree, that is, they all have the same distance from the root node. Figure 3.1 illustrates an example of an unchained $B+$-tree. In this figure, and in the remainder of this chapter, for simplicity, we refer to the content of each node with a label (e.g., a), instead of explicitly reporting the values in it. In the example, root node r has six children $(a, \ldots, f)$, each with three to four children. For easy reference, we label the children of a non-root node with the same label as the node concatenated with a progressive number (e.g., a_1, a_2, a_3 are the children of node a). At the *logical* level, each node is allocated to a logical identifier. Logical node identifiers are also used in internal nodes as pointers to their children. At the *physical* level, each node is translated into an encrypted chunk stored at a physical block. The encrypted chunk is obtained by encrypting the concatenation of the node identifier and its content (values and pointers to children). Encryption protects the confidentiality of nodes' content and of the tree structure. Also, it provides integrity of each node (as tampering would be detected) and of the structure overall (being the node identifier and the pointers to children also encrypted in the block).

Retrieval of a value in the tree requires walking the tree from the root to the target leaf, following at each level the pointer to the child in the path to the target leaf. Being the index stored in encrypted form, such an access requires an iterative process with the client downloading at each level (starting from the root) the block of interest, decrypting it, and determining the next block to be requested.

Although the data structure is encrypted, by observing a long enough sequence of accesses, the provider (or other observers able to see the accesses) could reconstruct the topology of the tree, identify repeated accesses, and possibly also infer sensitive data content [IKK14, PZM13].

To protect data and accesses from such inferences, the shuffle index makes use of the following three complementary techniques bringing confusion to the observer and destroying the static correspondence between nodes and blocks storing them.

- *Cover searches*: in addition to the target value, additional values, called *covers*, are requested. Covers, chosen in such a way to be indistinguishable from the target and to operate on disjoint – apart from the root – paths in the tree (also disjoint from the path of the target), provide uncertainty to the provider on the actual target. If *num_cover* searches are used, the provider will observe access to *num_cover*+1 distinct paths and corresponding leaf blocks, any of which could be the actual target.

- *Cache*: to avoid the provider learning when two accesses refer to the same target since they would have a path in common, the shuffle index avoids intersections between subsequent searches by locally storing at the client side a (plaintext) copy of the last *num_cache* visited paths to target nodes. The cache keeps track of only actual searches and is managed according to the Least Recently Used policy. If the target of an access already belongs to the local cache, *num_cover*+1 covers are used in the access.

- *Shuffling*: at every access, the nodes involved in the access are shuffled (i.e., allocated to different logical identifiers and corresponding physical blocks), re-encrypted (with a different random salt and including the new identifier of the block) and re-stored at the provider. Shuffling provides dynamic reallocation of all the accessed and cached nodes, thus destroying the otherwise static correspondence between physical blocks and their content. This prevents the provider from accumulating knowledge on the data allocation as at any access such an allocation is refreshed.

3.4 Rationale of the Approach

The approach presented in this chapter builds on the shuffle index by borrowing from it the base data structure (encrypted unchained $B+$-tree) and the idea of breaking the otherwise static correspondence between nodes and physical blocks at every access. It differs from the shuffle index in the management of the data structure, for both storage and access (which exploit a distributed allocation) and in the way the node-block correspondence is modified, applying swapping instead of random shuffling, forcing the node involved in an access to change the block storing it (again exploiting the distributed allocation). Also, it departs from the cache, not requiring any storage at the client side.

The basic idea of our approach is to randomly partition data among three independent Cloud providers, and, at every access, randomly move (*swap*) data retrieved from a provider to any of the other two so that data retrieved from a provider would not be at the same provider after the access. Since nodes are randomly allocated to providers, the path from the root to the leaf target of an access can traverse nodes allocated at different providers. Since each provider should operate as if it was the only one serving the client, our approach has to ensure *uniform visibility* at any access at every provider. Uniform visibility is obtained by requesting, every time a node is accessed at a given level at a provider, to access one additional block (distributed cover) at the same level at each of the other providers.

The main characteristics of our approach are then the use of swapping (in contrast to shuffling) and the use of three providers for managing the data structure.

The reader may wonder why we are distributing the index structure among *three* providers, and not two or four. The rationale behind the use of multiple providers is to provide limited visibility, at each of the providers, of the data structure and of the accesses to it. In this respect, even adopting two providers could work. However, an approach using only two providers would remain exposed to collusion between them. In fact, the two providers, by merging their knowledge, could reconstruct the node-block correspondence and compromise access and data confidentiality. In fact, collusion, while unlikely, cannot be completely ruled out (or in any case some protection must be provided against it). The data swapping we adopt, while providing better protection with respect to shuffling in general, implies deterministic reallocation in the case of two providers and could then cause exposure in case of collusion. Indeed, swapping forces an accessed node to be reallocated to a block at a different provider from the one where the node was stored before the access. The use of three then providers provides considerably better protection than the use of two. Swapping ensures that data move from a provider to another at every access, providing non-determinism in data reallocation (as the data could have moved to any of the other two providers), even in presence of collusion among the three providers. While going from two providers to three providers offers considerably higher protection guarantees, further increasing the number of providers offers limited advantage, while instead increasing the complexity of the system.

3.5 Data Structure and Three-Provider Allocation

At the abstract level, our structure is essentially the same as the shuffle index, namely we consider an unchained $B+$-tree defined over candidate key K, with fan-out F, and storing data in its leaves. However, we consider the root to have three times the capacity of internal nodes. Since internal nodes and leaves will be distributed to three different providers, assuming a three times larger root allows us to conveniently split it among the different providers (instead of replicating it) providing better access performance by potentially reducing the height of the tree. In fact, a $B+$-tree having at most $3F$ children for the root node can store up to three times the number of tuples/values stored in a traditional $B+$-tree of the same height. Formally, each internal abstract node n^a in the tree stores a list $v_1,\ldots,v_q$ of q values, with $\lceil \frac{F}{2} \rceil - 1 \leq q \leq F-1$ ($q \leq 3F-1$ for the root node), ordered from the smallest to the greatest, and has $q+1$ children. The i-th child of a node is the root of the subtree containing the values val with $v_{i-1} \leq val < v_i$, $i = 2,\ldots,q$; the first child is the root of the subtree with all values $val < v_1$, while the last child is the root of the subtree with all values $val \geq v_q$. Each leaf node stores a set of values, together with the tuples in the dataset having these values for attribute K. All the non-root nodes have to be at least 33% full. Figure 3.2(a) illustrates an example of our abstract data structure.

At the logical level, the abstract root node translates to three logical nodes, say r_0, r_1, r_2, each storing one third of the values and pointers to children of the abstract root node. More precisely, r_0 stores values $v_1,\ldots,v_i$, with $i = \lfloor \frac{q-2}{3} \rfloor$, and the corresponding pointers to children; r_1 stores values $v_{i+2},\ldots,v_{2i+1}$, and the corresponding pointers to children; and r_2 stores the remaining values $v_{2i+3},\ldots,v_q$, and the corresponding pointers to children. (Note that values v_{i+1} and v_{2i+2} are not necessary for the index definition and are then not explicitly stored in the obtained roots.) Figure 3.2(b) illustrates an example of logical index representing the abstract index in Figure 3.2(a) where the abstract root r is represented by three logical nodes, r_0, r_1, r_2, each having two of the six children of r. Each (non-root) abstract node n^a translates to a logical node n and is allocated to a logical identifier $n.id$, used also to represent the pointer to n in its parent. To regulate data distribution at the different providers, we distinguish three subsets ID_i, $i \in \{Y,G,B\}$, of logical identifiers

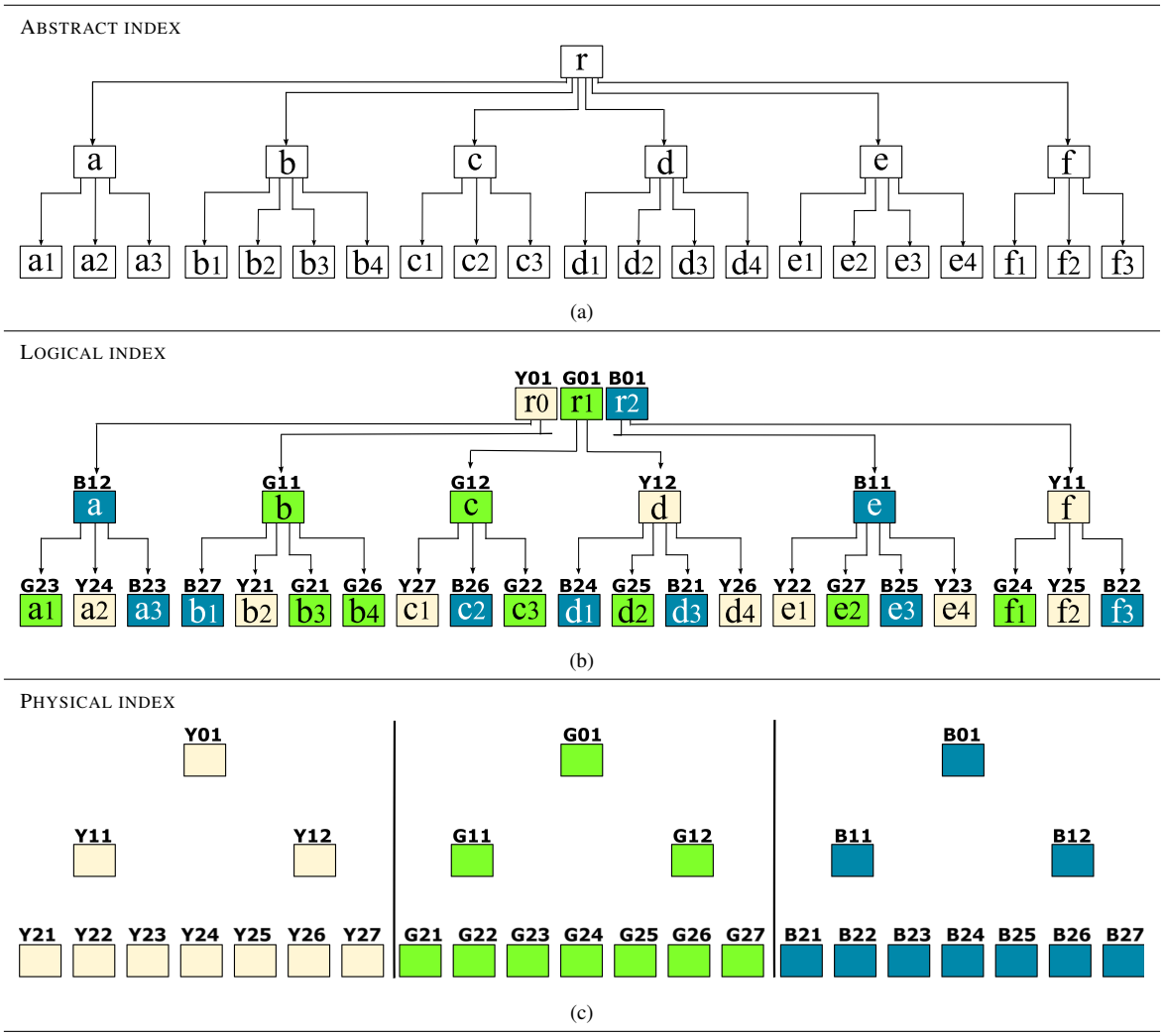


Figure 3.2: An example of abstract (a) and logical (b) index together with its physical representation (c) at three providers

corresponding to the physical addresses at each of the storage providers S_i, $i \in \{Y, G, B\}$. Allocation of abstract nodes to logical identifiers is defined through an allocation function, formally defined as follows.

Definition 3.5.1 (Distributed allocation)**.** *Let $\mathcal{N}^a$ be the set of abstract nodes in a distributed index I, S_Y, S_G, S_B be the providers storing I, and ID_Y, ID_G, ID_B be the sets of logical identifiers at provider S_Y, S_G, S_B, respectively. A* distributed allocation function *is a bijective function ϕ: $\mathcal{N}^a \rightarrow ID_Y \cup ID_G \cup ID_B$ that associates a logical identifier with each abstract node.*

Given an abstract node n^a, $\phi(n^a)$ determines the identifier of the logical node n where n^a is allocated, denoted $n.id$. In the following, we denote with $\sigma(id)$ the provider at which the logical node with identifier id is stored. Note that the order of logical identifiers is independent from the node content. Also, the allocation of logical nodes to physical blocks and, more in general, to providers does not depend on the topology of the abstract structure. In other words, a node may be stored at a different provider with respect to its parent and/or its siblings. An example of distribution of the index in Figure 3.2(a) is illustrated in Figure 3.2(b). For the sake of readability,

logical identifiers are reported on the top of each node and blocks are color-coded (yellow for S_Y, green for S_G, and blue for S_B, corresponding to light, medium, and dark gray in b/w printout). For simplicity and easy reference, logical identifiers start with a letter denoting the provider where the corresponding block is stored (Y for S_Y, G for S_G, and B for S_B), and their first digit denotes its level in the tree. For instance, G_{24} is the logical identifier of a node at level 2 and stored at provider S_G.

A distributed index I can be represented, at the logical level, as a pair $\langle \mathcal{N}, (S_Y,S_G,S_B)\rangle$, with $\mathcal{N}$ the set of logical nodes composing it, and S_Y, S_G, and S_B the storage providers where these nodes are physically stored. To guarantee distribution among the different providers (and provide uniform visibility at every provider in access execution), the distributed allocation function guarantees that at level 1 (children of the root) there is at least one node stored at each storage provider, and that each non-root node in the index has at least one child stored at each provider. At starting time, we then assume the structure to be evenly distributed at the level of node, meaning that the children of each node are equally distributed among the three providers (i.e., each provider will be allocated one third ± 1 of the children of every node). We also assume the structure to be evenly distributed both globally and for each level in the tree. Figure 3.2(b) represents an example of logical distributed index where the children of each node, the nodes in each level, and the nodes in the tree are evenly distributed to providers.

At the physical level, logical addresses are translated into physical addresses at the three providers. Node content is prefixed with a random salt and encrypted in Cipher Block Chaining (CBC) mode with a symmetric encryption function. The result of encryption is concatenated with the result of a Message Authentication Code (MAC) function applied to the encrypted node and its identifier, producing an encrypted block b allocated to a physical address. Formally, the block b representing a node n is the concatenation $\mathcal{E}||\mathcal{T}$ of two strings obtained as follows: $\mathcal{E} = E_{k_1}(salt||n)$ and $\mathcal{T} = MAC_{k_2}(id||\mathcal{E})$, with E a symmetric encryption function, k_1 the encryption key, MAC a strongly unforgeable keyed cryptographic hash function with key k_2, and $salt$ a randomly chosen value. The presence of the node identifier in each block enables the client to assess the authenticity and integrity of the block content and, thanks to the identifiers of the children stored in each node, also of the whole index structure. Figure 3.2(c) illustrates the physical representation of the logical index in Figure 3.2(b). In the following, for simplicity and without loss of generality, we assume that the physical address of a block corresponds to the logical identifier of the node it stores. The view of each provider S_i, with $i \in \{Y,G,B\}$, corresponds to the portion of the physical representation in Figure 3.2(c) allocated at S_i. Note that each provider can see all and only the blocks allocated to it. In the following, we use the term node to refer to an abstract data content and block to refer to a specific memory slot in the logical/physical structure. When either terms can be used, we will use them interchangeably.

3.6 Working of the Approach

In this section, we illustrate how access execution is performed adopting *distributed covers* and *swapping* protection techniques to guarantee data and access confidentiality.

3.6.1 Distributed Covers

Like in the shuffle index, retrieval of a key value (or more precisely the data indexed with that key value and stored in a leaf node) entails traversing the index starting from the root and following,

at every node, the pointer to the child in the path to the leaf possibly containing the target value. Again, since data are encrypted, such a process needs to be performed iteratively, starting from the root to the leaf, at every level decrypting (and checking integrity of) the retrieved node to determine the child to follow at the next level. Since our data structure is distributed among three providers and the allocation of nodes to providers is independent from the topology of the index structure, the path from the root to a target leaf may (and usually does) involve nodes stored at different providers. For instance, with reference to Figure 3.2, retrieval of a value d_1 entails traversing path $\langle r_1, d, d_1 \rangle$ and hence accessing blocks G_{01}, Y_{12}, and B_{24} each stored at a different provider. Retrieval of value a_3 entails traversing the path $\langle r_0, a, a_3 \rangle$ and hence accessing blocks Y_{01}, B_{12}, and B_{23}, the first stored at provider S_Y and the last two stored at provider S_B. Since each provider can observe different iterations and, after a long enough sequence of observations, also infer the levels associated with blocks, we aim at ensuring a uniform visibility at every provider and at each access. In other words, we want every provider to observe, for every search, the access to one block at each level, with each provider then operating as if it was the only one serving the client. This approach guarantees that each provider has uniform visibility over every access, independently from the allocation of the target of the search. (Note that even if only one block is accessed at every level, no information is leaked to the provider on the tree topology, since: *i)* the accessed blocks may not be actually in a parent-child relationship, and *ii)* the content of accessed blocks changes just after the access.) Our requirement of uniform visibility at each provider is captured by the following property.

Property 3.6.1 (Uniform visibility)**.** *Let $I = \langle \mathcal{N}, (S_Y, S_G, S_B) \rangle$ be a distributed index, and $N = \{n_1, \ldots, n_m\}$ be the set of logical nodes accessed by a search. The search satisfies* uniform visibility *iff for each S_i, $i \in \{Y, G, B\}$, and for each level l in I, $\exists!\ n \in N$ such that:* 1) $\sigma(n.id) = S_i$*; and* 2) *n is at level l in I.*

In other words, for each access, one and only one node per level should be accessed at every provider. To illustrate, our two sample accesses above do not satisfy uniform visibility. For instance, in the first access, provider S_G is accessed for level 0 (G_{01}), but not for levels 1 and 2. To satisfy uniform visibility, we complement, at each level, the access required by the retrieval of the target value with two additional accesses at the providers that do not store the target block at that level. We call *covers* these additional accesses as they resemble cover searches of the shuffle index, although they have also many differences (e.g., they cannot be pre-determined as data allocation is unknown, they may not represent a path in the distributed index, and they are not observed by the same provider observing the target). Stressing their distributed nature, we term them distributed covers, defined as follows.

Definition 3.6.1 (Distributed cover)**.** *Let $I = \langle \mathcal{N}, (S_Y, S_G, S_B) \rangle$ be a distributed index, and n be a node in $\mathcal{N}$. A set of* distributed covers *for n is a pair of nodes (n_i, n_j) in $\mathcal{N}$ such that the following conditions hold:* 1) *n, n_i, n_j belong to the same level of I; and* 2) *$\sigma(n.id) \neq \sigma(n_i.id)$, $\sigma(n.id) \neq \sigma(n_j.id)$, and $\sigma(n_i.id) \neq \sigma(n_j.id)$.*

As stated by the definition above, distributed covers for a node n are a pair of nodes (n_i, n_j) that belong to the same level l of the structure as n, and such that the three nodes are allocated at different providers. For instance, distributed covers for Y_{12} could be any of the following pairs: (B_{11}, G_{11}), (B_{11}, G_{12}), (B_{12}, G_{11}), (B_{12}, G_{12}). Similarly, at the leaf level, the distributed covers for B_{24} could be any pair of nodes (Y_{2*}, G_{2*}), with $*$ any value between 1 and 7 (e.g., (Y_{23}, G_{21})). The

distributed covers of a root node are the roots at the other two providers (e.g., (G_{01},B_{01}) are the distributed covers for Y_{01}).

With the consideration of distributed covers, to guarantee uniform visibility at every provider, access execution works as follows. Again, an iterative process is executed starting from the root to the leaf level. First, the client retrieves the roots at all the three providers and decrypts them to determine the target root (i.e., the one going to the target value) and the target child node n to visit. It also randomly chooses two distributed covers for n. The client requests access to n and its distributed covers to the respective providers. It then decrypts the accessed nodes and iteratively performs the same process until the leaves (target and distributed covers) are reached. As an example, consider the data structure in Figure 3.2(b) and assume node d_1 to be the target. The nodes along the path to the target of the accesses are $\langle r_1,d,d_1\rangle$ entailing accesses to target blocks $\langle G_{01},Y_{12},B_{24}\rangle$. Assume that distributed covers (Y_{01},B_{01}), (G_{11},B_{11}), and (G_{21},Y_{23}) are used for G_{01}, Y_{12}, and B_{24}, respectively. Figure 3.3(a) illustrates the nodes involved in the access, either as target (denoted with a bullet) or as distributed covers, at each level also indicating the parent-child relationship among them at the abstract level. Figure 3.3(b) provides the same information distinguishing the nodes accessed at every provider. Note that each provider simply observes a sequence of three accesses to three blocks, while the node content (reported in the figure for clarity) is not visible to the providers.

Even if any pair of nodes at the same level as n, but allocated at the other two providers, can work as distributed covers for n, in the choice of distributed covers we need to take into consideration the fact that accessed nodes are reallocated. In fact, when n is moved to a different block, the pointers to n in its parent must be updated to maintain consistency of the index structure. Therefore, the nodes involved in an access should always form a sub-tree, possibly including paths of different lengths. Each distributed cover at level l should then be child of the node along the path to the target at level $l-1$ or of one of its distributed covers. This is formally captured by the following definition.

Definition 3.6.2 (Chained distributed covers)**.** *Let $p=\{n_0,\ldots,n_h\}$ be the path (sequence of nodes) to the target, and $\mathcal{C}(p)=\langle(n_{0i},n_{0j}),\ldots,(n_{hi},n_{hj})\rangle$ be the sequence of distributed covers for the nodes in the path. $\mathcal{C}(p)$ is* chained *if $\forall x=1,\ldots,h$, n_{xi} and n_{xj} are children of one of the nodes in $\{n_{x-1},n_{(x-1)i},n_{(x-1)j}\}$.*

In other words, every node in $\mathcal{C}(p)$, except for the roots, must have its parent in $\mathcal{C}(p)$. The distributed covers in Figure 3.3(a) are chained as the covers at every level are children of a node accessed (either as target or cover) in the level above. Note that while in the example (for simplicity and readability of the figure) every accessed node has one accessed child, such a condition is not needed. In fact, Definition 3.6.2 requires every node to have its parent in the access (so to enable update of pointers to the node in its parent), while a node can have no children in the access. The reason for such a choice is twofold: it provides better protection to the parent-child relationship among accessed nodes, and it permits to find more easily distributed covers for target nodes. For instance, Y_{26} (d_4) could have also been used instead of Y_{23} (e_4) as one of the covers for B_{24}, together with G_{21} still satisfying Definition 3.6.2.

3.6.2 Swapping

A desired requirement of our approach is that data retrieved (either as target or as cover) in an access are stored after the access at a different provider. We capture such a requirement with a property of *continuous moving* as follows.

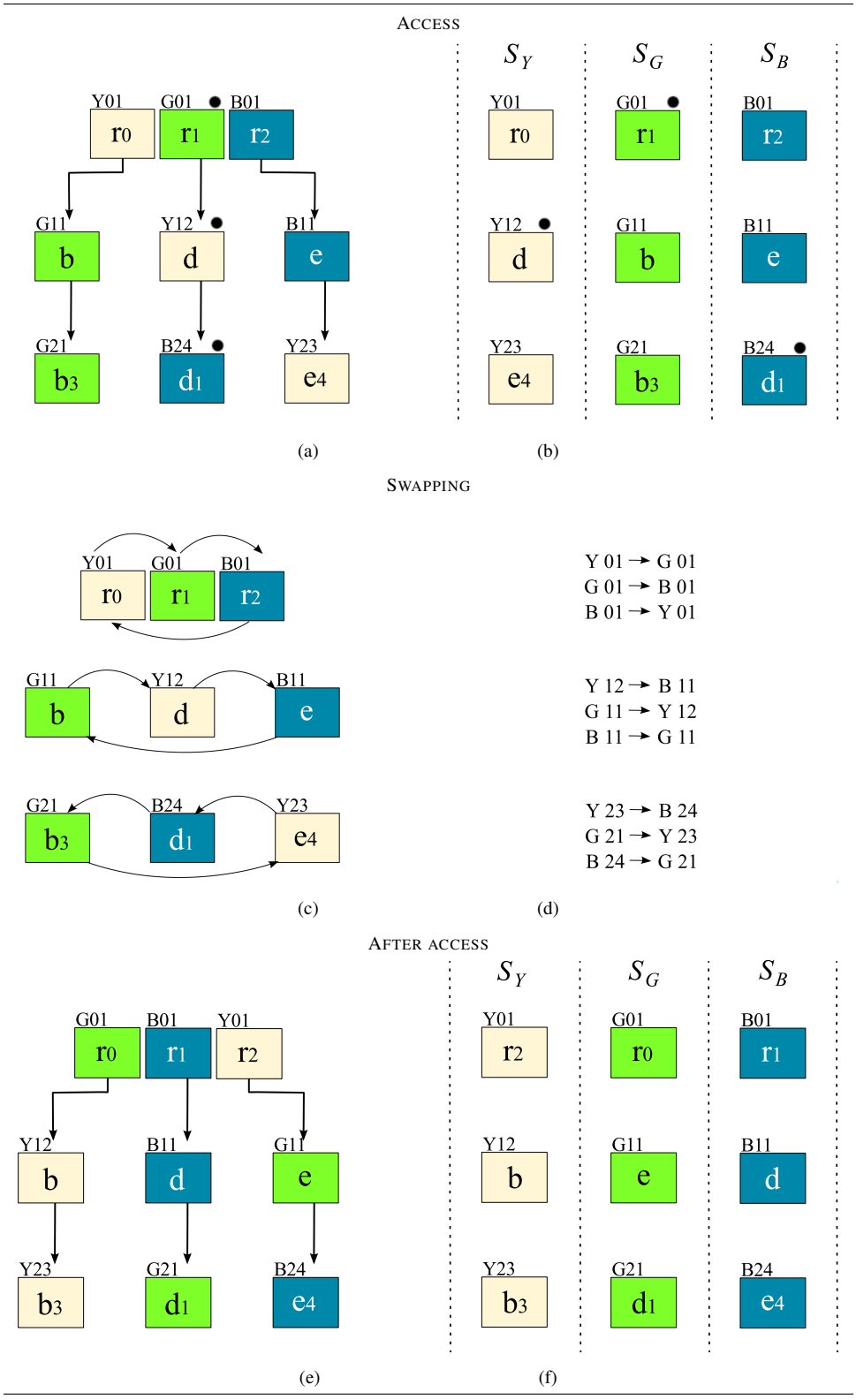


Figure 3.3: Logical (a) and physical (b) view of the nodes/blocks accessed searching for value d_1; swapping among accessed nodes/blocks (c-d); logical (e) and physical (f) view of the effects of the swapping. Target nodes/blocks are denoted with •

Property 3.6.2 (Continuous moving)**.** *Let $I = \langle \mathcal{N}, (S_Y, S_G, S_B) \rangle$ be a distributed index, and $N = \{n_1, \ldots, n_m\}$ be the set of nodes in $\mathcal{N}$ accessed as target or distributed covers by a search. The search satisfies* continuous moving *iff, for each node $n \in N$, the provider $\sigma(n.id)$ where n is stored before the access is different from the one where it is stored after the access.*

Continuous moving prevents providers from building knowledge based on accesses they can observe, since a node is immediately removed from a provider after being accessed. For instance, providers will not be able to observe repeated accesses anymore. We guarantee satisfaction of this property by swapping the content of the blocks accessed at every level. Swapping is defined as follows.

Definition 3.6.3 (Swapping)**.** *Let ID be a set of logical identifiers. A* swapping *for ID is a random permutation $\pi : ID \rightarrow ID$ such that $\forall id \in ID$, $\sigma(id) \neq \sigma(\pi(id))$.*

Figure 3.3(d) illustrates a possible swapping among the nodes/blocks accessed at each provider by the search in Figure 3.3(a-b), resulting in swapping content among them as depicted in Figure 3.3(c). For instance, swap $Y_{01} \rightarrow G_{01}$, $G_{01} \rightarrow B_{01}$, $B_{01} \rightarrow Y_{01}$ causes r_0 to move to G_{01}, r_1 to move to B_{01}, and r_2 to move to Y_{01}. Figure 3.3(e) illustrates the effect of such a swapping on the data structure at the logical level. Figure 3.3(f) shows the changes in the content of blocks stored at each provider. Note that before re-writing blocks at the providers, the content of the corresponding nodes is re-encrypted with a different random salt that changes at every access. The adoption of a different random salt in node encryption and the concatenation with a different node identifier guarantees to produce a different encrypted block, even if the content represents the same node. This makes it impossible for storage providers to track swapping operations. Given an index characterized by a distributed allocation function ϕ and a swapping function π over a subset ID of the identifiers in the distributed index, the allocation function resulting from the swap is defined as: $\phi(n^a) = \pi(\phi(n^a))$ iff $\phi(n^a) \in ID$; $\phi(n^a) = \phi(n^a)$, otherwise. Note that the assignment function resulting from the application of a swap π still represents a distributed assignment function, since π is a permutation function. For instance, with reference to the example in Figure 3.3, we note that each node is associated with exactly one identifier and vice versa before and after the access. Figure 3.4 illustrates the logical distributed index before and after the access searching for d_1 and the swapping among accessed nodes, which preserves the correctness of the allocation function (Definition 3.5.1).

Moving nodes among providers may reduce the number of children at a provider for some nodes. In the worst case, a node may be left with no children on one of the three providers. We note however that, since we initially define a balanced allocation and in traditional systems the fan-out of the tree is high (in the order of some hundreds), the probability that a node is left without children on one of the providers is extremely low, due to a natural regression to the mean that reduces the stochastic drift. To completely solve this risk, as illustrated in the next section, we check that swapping does not create configurations where a provider is not represented in the descendants of a node.

3.6.3 Access Execution Algorithm

Figure 3.5 illustrates the pseudocode of the algorithm, executed at the client-side, searching for a value in our approach. The algorithm visits the index level by level, starting from the root. At each level l the algorithm chooses two distributed covers for the node along the path to the

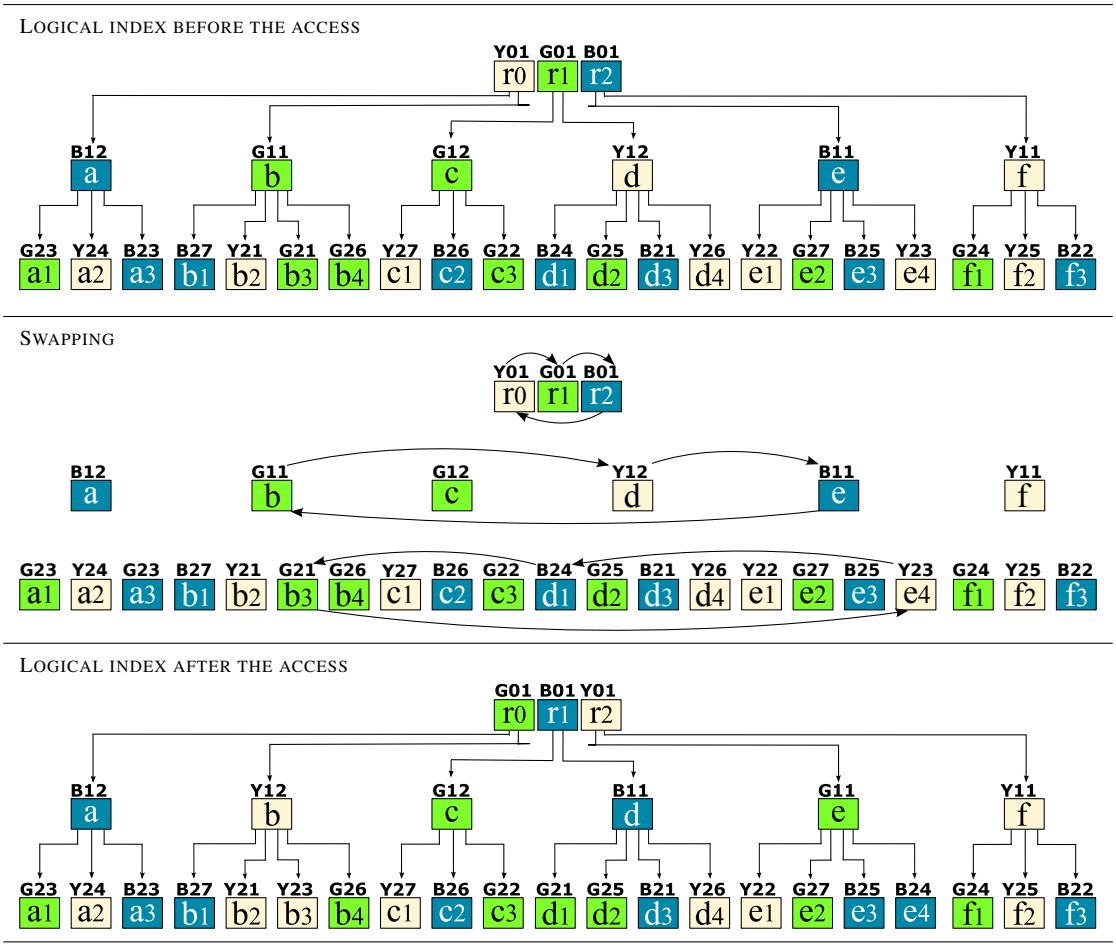


Figure 3.4: Evolution of the logical index in Figure 3.2 for the search of d_1 illustrated in Figure 3.3

target, accesses the target and cover blocks, decrypt them, and swap their content. To guarantee consistency of the index, swapping is also reported in the parents of accessed nodes, which are then re-encrypted and re-written at the storage providers. Finally, the algorithm returns the leaf node storing the target value.

3.7 Protection Analysis

We evaluate the guaranteed offered by our approach with respect to guaranteeing confidentiality of the accesses against possible observers. In particular, we consider the providers as our observers as they have the most powerful view over the stored data as well as of the accesses to them (Section 3.7.1). Guaranteeing confidentiality of the accesses means hiding from the providers the correspondence (as our distribution and swap aim to do) between nodes and blocks where they are stored. We perform our analysis considering two different, opposite, starting scenarios. The first one represents a worst case scenario where, at initialization time, each provider knows the node-block correspondence exactly (Section 3.7.2). We then illustrate how our approach is able to quickly destroy the knowledge of the providers at every access, even in presence of collusion among them. The second scenario considers instead the case where the providers do not have any knowledge at initialization time about node-block correspondence (Section 3.7.3). We then

/* $I = \langle \mathcal{N}, (S_Y, S_G, S_B) \rangle$: distributed index with height h */
INPUT *target_value* : value to be searched in I
OUTPUT n : leaf node that contains *target_value*
MAIN
1: *Parents* := download and decrypt block Y_{01} from S_Y, block G_{01} from S_G, and block B_{01} from S_B
2: let π be a permutation of identifiers of nodes in *Parents* s.t. $\sigma(id) \neq \sigma(\pi(id))$
3: swap nodes in *Parents* according to π
4: **for** l:=1...h **do** /* visit the index level by level*/
5: *target_id* := id of the node at level l along the path to *target_value*
6: randomly choose *cover*[1] and *cover*[2] s.t. they are children of *Parents* and
$\sigma(target\_id) \neq \sigma(cover[1])$, $\sigma(target\_id) \neq \sigma(cover[2])$, and $\sigma(cover[1]) \neq \sigma(cover[2])$
7: *Read* := download and decrypt each block with identifier $id \in \{target\_id, cover[1], cover[2]\}$ from $\sigma(id)$
8: let π be a permutation of identifiers of nodes in *Read*
s.t. $\sigma(id) \neq \sigma(\pi(id))$ and each $n \in Parents$ has at least one child at S_Y, S_G, S_B
9: **if** π does not exist, **then** goto 6
10: swap nodes in *Read* according to π
11: update pointers to children in *Parents* according to π
12: encrypt and write each node $n \in Parents$ at provider $\sigma(n.id)$
13: *target_id* := π(*target_id*)
14: *cover*[1] := π(*cover*[1]), *cover*[2] := π(*cover*[2])
15: *Parents* := *Read*
16: encrypt and write each node $n \in Parents$ at provider $\sigma(n.id)$
17: return node $n \in Read$ with $n.id = target\_id$

Figure 3.5: Access algorithm

illustrate how our approach prevents the providers from building knowledge on the node-block correspondence based on their knowledge on the accesses being performed.

3.7.1 Modeling Knowledge

The storage providers know (or can infer from their interactions with the client) the following information: the total number of blocks (nodes) in the distributed index; the height h of the tree structure; the identifier of each block b and its level in the tree; the identifier of read and written blocks for each access operation. On the contrary, they do not know nor can infer the content and the topology of the index (i.e., the pointers between parent and children), thanks to the fact that nodes are encrypted. For simplicity, but without loss of generality, we focus our analysis only on leaf blocks/nodes, since leaves are considerably more exposed than internal nodes. Internal nodes are more protected since they are accessed, and hence involved in swapping operations, more often than leaf nodes.

Let $\mathcal{N}$ be the set of logical leaf nodes in the unchained $B+$-tree, and $\mathcal{B}$ be the set of blocks storing them at any of the providers. We assume data to be equally distributed among storage providers S_Y, S_G, and S_B (i.e., the number N of nodes stored at each provider is equal to $\frac{|\mathcal{N}|}{3}$). For concreteness and simplicity of notation, we assume that $|\mathcal{N}|$ is a multiple of 3, and that blocks $b_1, \ldots, b_N$ are at S_Y, blocks $b_{N+1}, \ldots, b_{2N}$ are at S_G, and blocks $b_{2N+1}, \ldots, b_{3N}$ are at S_B.

The knowledge of a provider S_x, with $x \in \{Y, G, B\}$, on the fact that a node n is stored at a block b can be expressed as the probability $P_x(b,n)$ that S_x knows that node n is stored at block b. Probability $P_x(b,n)$ has value 1 if the provider knows with certainty that n is stored at b, and value $\frac{1}{|\mathcal{N}|}$ for every n if the provider does not have any knowledge on node-block allocation (i.e., the

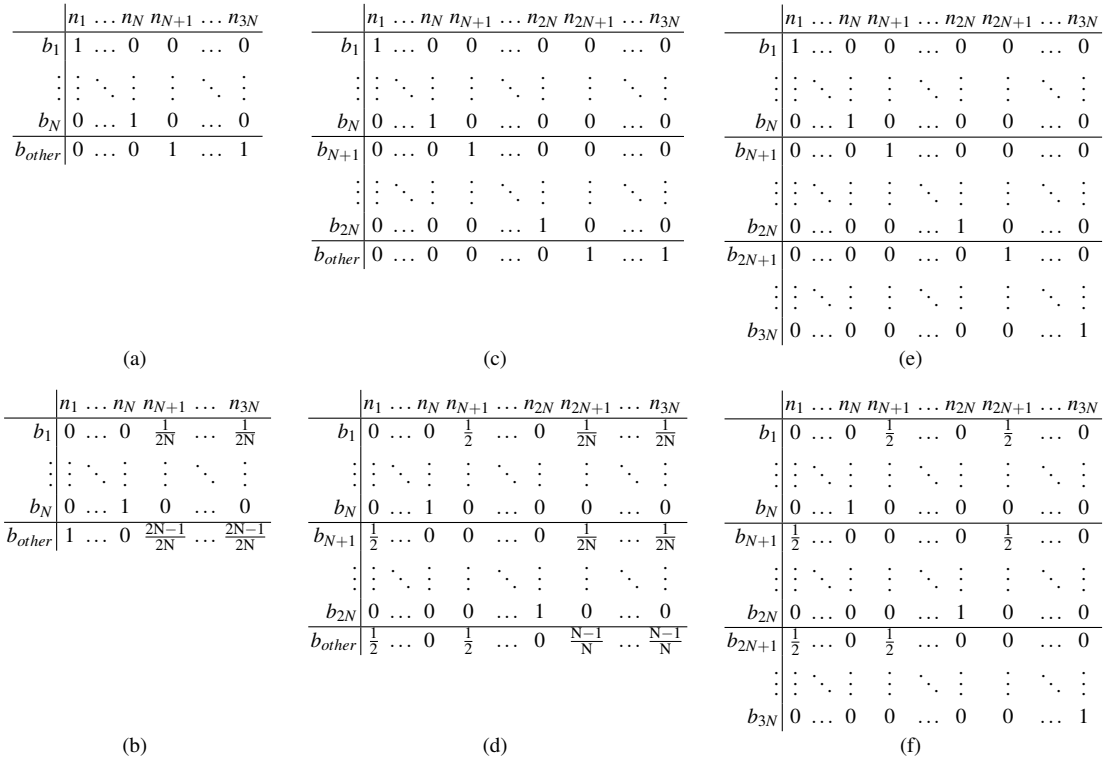


Figure 3.6: Probability matrices in the worst case scenario at initialization (a,c,e) and after the first access (b,d,f) to blocks b_1, b_{N+1}, b_{2N+1} with: no collusion (a,b), collusion among two providers (c,d), and full collusion (e,f)

block could contain n or any other node in $\mathcal{N}$). The overall degree of uncertainty of a provider S_x about the block containing a node n can be represented as the entropy H_n^x, computed on the non-zero probabilities $P_x(b_i,n)$, for all $b_i \in \mathcal{B}$, that is, $H_n^x = -\sum_{i=1}^{|\mathcal{B}|} P_x(b_i,n) \cdot (\log_2 P_x(b_i,n))$. Note that $H_n^x = 0$ means that the provider knows exactly the block storing n. In fact, in this case $P_x(b_i,n) = 1$ for each block b_i and then $H_n^x = -\sum_{i=1}^{|\mathcal{B}|} 1 \cdot (\log_2 1) = 0$. On the contrary, $H_n^x = \log_2 |\mathcal{N}|$ means that the provider has complete uncertainty about such a correspondence. In fact, $P_x(b_i,n) = \frac{1}{|\mathcal{N}|}$ for each block b_i and then $H_n^x = -\sum_{i=1}^{|\mathcal{B}|} \frac{1}{|\mathcal{N}|} \cdot (\log_2 \frac{1}{|\mathcal{N}|}) = \log_2 |\mathcal{N}|$.

3.7.2 Knowledge Degradation

In the worst case initialization scenario, every provider S_x, with $x \in \{Y,G,B\}$, initially knows the exact correspondence between nodes and its blocks (i.e., $H_n^x = 0$ since $P_x(b,n) = 1$ if n is allocated at b, $P_x(b,n) = 0$ otherwise, with n a node in $\mathcal{N}$ and b one of the blocks stored at S_x). We show, for each node n, the evolution of entropy H_n^x (i.e., knowledge degradation) for each provider as a consequence of a random sequence of accesses. For concreteness, we refer the discussion to provider S_Y (but the same applies to S_G and S_B) and write $P(b,n)$ instead of $P_Y(b,n)$ and H_n instead of H_n^Y.

No collusion. We first consider the natural configuration where providers do not collude, that is, each provider has knowledge of the overall sets of nodes but observes only the encrypted content and accesses to the blocks it stores. At initialization time, S_Y has complete knowledge on the node-block correspondence for the blocks it stores, while it does not have any information on

the allocation of nodes to blocks stored at the other two providers. For simplicity, we assume each node n_i to be initially allocated at block b_i (i.e., $P(b_i,n_i)$=1 and $P(b_i,n_j)$=0). Figure 3.6(a) illustrates the probability values for the different nodes and blocks: each cell $[b_i,n_j]$ in the matrix reports the value of $P(b_i,n_j)$. Consistently with the fact that S_Y does not have any information on blocks at the other providers, all such blocks are summarized in a single row b_{other} in the matrix, which reports the probability that n_j is not at S_Y (i.e., b_{other} is the sum of the probabilities $P(b_i,n_j)$ with b_i a block not at S_Y). We now illustrate how such probability values (and then the entropy) evolve as accesses are executed. In the discussion, we use $\mathbf{b}_i$ to denote the whole row in the probability matrix associated with block b_i, that is, the vector over cells $[b_i,n_j]$, with $j = 1,\ldots,3N$. Operations over rows are to be interpreted to operate cell-wise.

Consider the first access observed by S_Y and let b_y (storing n_y) be the block accessed. Because of swapping, after the access, block b_y will certainly not contain anymore node n_y since the node has moved to one of the blocks at the other two providers (with equal probability among the blocks, as S_Y does not have any knowledge over them). Hence, $P(b_y,n_y)$ will change to 0 (from 1 before the access) while $P(b_{other},n_y) = 1$ (from 0 before the access). Similarly, b_y could contain after the access, with equal probability, any of the other $2N$ nodes previously stored at one of the other providers. For any n_j with $j > N$, $P(b_y,n_j)$ will change to $\frac{1}{2N}$ (from 0 before the access) and $P(b_{other},n_j)$ will change to $\frac{2N-1}{2N}$ (from 1 before the access). Assuming, as an example, that the first access is for block b_1, Figure 3.6(b) illustrates the probability matrix after the access execution. Extending the reasoning to a sequence of accesses, we can formalize the changes to the probability matrix due to the observation of the access to a block b_y as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{2N}\mathbf{b}_{other}$;
- $\mathbf{b}_{other} := \mathbf{b}_y + \frac{2N-1}{2N}\mathbf{b}_{other}$;
- $\mathbf{b}_i := \mathbf{b}_i$, with $i \neq y$ and $i \neq other$.

Note that the first access described above is indeed an instantiation of these formulas. In fact, with reference to our example, $P(b_1,n_1)$ changes from 1 to $\frac{1}{2N} \cdot 0 = 0$ after the access.

To evaluate the increase of entropy due to changes in allocation probabilities, we performed a series of simulations with indexes of different sizes. In the simulations, we considered both uniform and non-uniform distributions of the logical access requests and this aspect did not have a detectable impact on the results. Figure 3.7 shows the results of the simulations and confirm that the initial knowledge of the provider suffers rapid degradation, independently from the index size. While large indexes show a less steep increase in the entropy trend, this is balanced by the greater uncertainty on the node-block correspondences due to their large number of leaf nodes/blocks.

Collusion between two providers. We now evaluate protection (i.e., destruction of knowledge for this scenario) in presence of collusion between two providers. For concreteness and easy notation, let us assume the two colluding providers to be S_Y and S_G. By colluding, S_Y and S_G combine their knowledge of the initial node-block correspondence, producing the initial probability matrix illustrated in Figure 3.6(c), where b_{other} now refers to the blocks at provider S_B, over which S_Y and S_G have no knowledge. Also, S_Y and S_G can combine their observations on the accesses. Assume then the initial configuration and a first observation on the access of block b_y (storing n_y at S_Y) and block b_g (storing n_g at S_G). Let us first consider block b_y and node n_y, as the same (just substituting g for y and vice versa) applies to b_g and n_g. Because of swapping, after the access, block b_y will certainly not contain anymore node n_y since the node content has been moved to

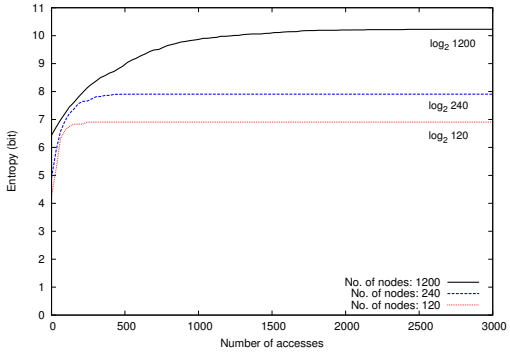
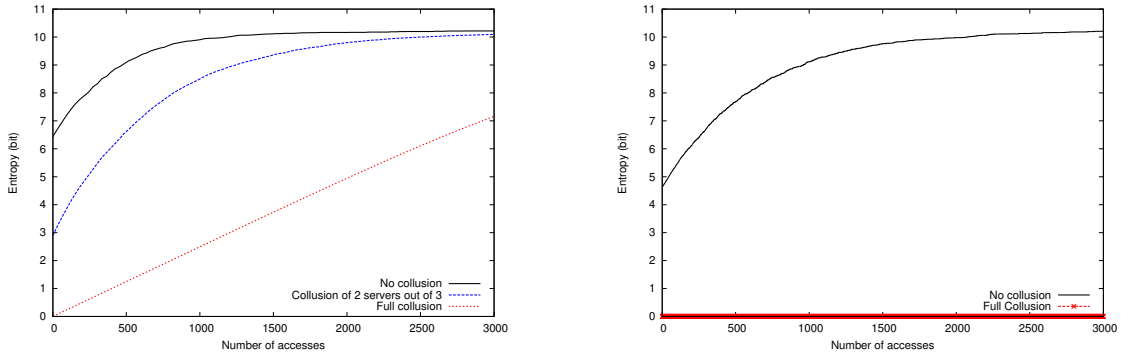


Figure 3.7: Entropy evolution varying the number of leaf nodes



(a) (b)

Figure 3.8: Entropy evolution for an index with 1200 leaf nodes distributed among three (a) and two (b) storage providers

either b_g (with probability $\frac{1}{2}$) or to any other (equiprobable) block at S_B (again with probability $\frac{1}{2}$). Hence, $P(b_y, n_y)$ will change to 0 (from 1 before the access) while $P(b_g, n_y) = P(b_{other}, n_y) = \frac{1}{2}$ (from 0 before the access). Similarly, b_y could contain after the access, either n_g (with probability $\frac{1}{2}$) or any of the nodes previously stored in b_{others} (each with equal probability $\frac{1}{2N}$). Formally, $P(b_y, n_g) = \frac{1}{2}$ (from 0 before the access) while for any n_j with $j > 2N$, $P(b_y, n_j)$, will change to $\frac{1}{2N}$ (from 0 before the access) and $P(b_{other}, n_j)$ will change to $\frac{N-1}{N}$ (from 1 before the access). Assume, as an example, that the first access is for block b_1 at S_Y and for block b_{N+1} at S_G, Figure 3.6(d) illustrates the probability matrix after the access execution. Extending the reasoning to a sequence of accesses, we can formalize the changes to the probability matrix due to the observation of the access to blocks b_y and b_g as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{2}(\mathbf{b}_g + \frac{1}{N}\mathbf{b}_{other})$;
- $\mathbf{b}_g := \frac{1}{2}(\mathbf{b}_y + \frac{1}{N}\mathbf{b}_{other})$;
- $\mathbf{b}_{other} := \frac{1}{2}(\mathbf{b}_y + \mathbf{b}_g) + \frac{N-1}{N}\mathbf{b}_{other}$;
- $\mathbf{b}_i := \mathbf{b}_i$, with $i \neq y$, $i \neq g$, and $i \neq other$.

We performed some simulations evaluating the evolution of entropy in presence of collusion

between two providers comparing it with the base case where no collusion exists. Figure 3.8(a) illustrates the results of such simulations for a data structure with $|\mathcal{N}| = 1200$, where the solid black line corresponds to the base case of no collusion and the dotted blue line (darker in b/w printout) to the case where two providers collude. We note that even if the entropy shows a less steep increase than in the scenario with no collusion (as it is to be expected given the combined knowledge of the two providers), our approach still provides considerable degradation in the knowledge of colluding providers. In fact, even colluding, the two providers still cannot detect whether an accessed node has been allocated to one of them or to any of the other blocks not under their control.

Full collusion. We now evaluate protection (i.e., destruction of knowledge) in presence of collusion among all the three providers. By colluding, the providers can share information on the initial node-block correspondence, which would then be completely known to them as shown in Figure 3.6(e), and on the block accessed. Assume then the initial configuration and a first observation on the access to blocks b_y (storing n_y at S_Y), b_g (storing n_g at S_G), and b_b (storing n_b at S_B). Again, let us first consider block b_y and node n_y, as the same applies to b_g and n_g, and to b_b and n_b. Because of swapping, after the access block b_y will certainly not contain anymore node n_y as the node has moved either to b_g or to b_b, each with probability $\frac{1}{2}$. Hence, $P(b_y, n_y)$ will change to 0 (from 1 before the access) while $P(b_g, n_y) = P(b_b, n_y)$ becomes $\frac{1}{2}$ (from 0 before the access). Similarly, b_y could contain after the access, either n_g or n_b each with probability $\frac{1}{2}$. Such changes in the probability are illustrated in Figure 3.6(f), which assumes, as an example, that the first access is for block b_1 at S_Y, b_{N+1} at S_G, and b_{2N+1} at S_B. Extending the reasoning to a sequence of accesses, we can formalize the changes to the probability matrix due to the observation of the access to blocks b_y, b_g, and b_b as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{2}(\mathbf{b}_g + \mathbf{b}_b)$;
- $\mathbf{b}_g := \frac{1}{2}(\mathbf{b}_y + \mathbf{b}_b)$;
- $\mathbf{b}_b := \frac{1}{2}(\mathbf{b}_y + \mathbf{b}_g)$;
- $\mathbf{b}_i := \mathbf{b}_i$, with $i \neq y$, $i \neq g$, and $i \neq b$.

In Figure 3.8(a), entropy evolution in case of full collusion is represented by the dotted red line (lighter in b/w printout). The increase of entropy (i.e., the knowledge degradation) is clearly lower than in the case of no or partial (between two providers) collusion. However, the knowledge of each provider is progressively destroyed thanks to the uncertainty (among the two other providers) of the new allocation of the accessed node.

3.7.3 Knowledge Gain

We now evaluate, with a similar approach, a different initialization scenario, where the providers have no knowledge on the node-block correspondence, and assume that a storage provider (or more of them in case of collusion) knows the content of the node n^* target of one every T accesses (e.g., the target of the 4th, 8th, 12th, ... accesses is known to the provider). However, it does not know the corresponding distributed covers, which are randomly chosen. To model the knowledge of the providers on the content of the target node (one every T access requests), in the analysis of the no collusion, collusion between two providers, and full collusion scenario, we select n^* uniformly at random among the leaves of the abstract index. The block storing n^* is, in turn, chosen following

the discrete probability mass function (pmf) specified by the column in the probability matrix associated with the target node n^*. The two blocks representing the distributed covers of n^* are chosen among the blocks stored at the other two providers. In the following, we show for each node n, the evolution of entropy H_n (i.e., knowledge gain) for each provider given the execution of a random sequence of accesses.

No collusion. We first consider the natural configuration where providers do not collude. At initialization time, S_Y has no knowledge on the node-block correspondence. Therefore, the initial configuration of the probability matrix is uniform (see Figure 3.9(a)). Consider the first access observed by S_Y to a known target node n^*, and let b_y be the block accessed at S_Y, and b_i any other block stored at S_Y. Since the target of the access is known to S_Y, it knows for sure that n^* is allocated (both before and after swapping) at one among the three accessed blocks with equal probability. Hence, the probability $P(b_y, n^*)$ will change to $\frac{1}{3}$ (from $\frac{1}{3N}$ before the access), while the probability that node n^* is allocated at a block b_i stored at S_Y different from b_y (and then not accessed) changes to $P(b_i, n^*) = 0$. The probability that n^* is allocated at a block stored at a provider different from S_Y instead remains unchanged $P(b_{other}, n^*) = \frac{2}{3}$. The values of the probabilities $P(b_y, n_j)$ that each non-target node $n_j \neq n^*$ is allocated at b_y before the access are uniformly redistributed over the whole set of $3N$ blocks, that is, $\frac{1}{3N}P(b_y, n_j)$ is added to $P(b_i, n_j)$, with $n_j \neq n^*$ and $i \neq y$. (In fact, it is more likely for blocks different from b_y to store nodes different from n^*.) Similar changes apply to the probability values for blocks that are stored at S_G and S_B. Hence, $2N \cdot \frac{1}{3N}P(b_y, n_j) = \frac{2}{3}P(b_y, n_j)$ is added to $P(b_{other}, n_j)$. Analogously, $P(b_y, n_j)$ is reduced to be $\frac{1}{3N}$ of its original value, that is $\frac{1}{3N}P(b_y, n_j)$. Figure 3.9(b) illustrates the probability matrix after the execution of an access with target n^* that accesses block b_1. Extending the reasoning to a sequence of accesses, we can formalize the changes to the probability matrix due to the observation of an access with known target n^* as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{3N}\mathbf{b}_y$ and $\mathbf{b}_y[n^*] := \frac{1}{3}$;
- $\mathbf{b}_{other} := \mathbf{b}_{other} + \frac{2}{3}\mathbf{b}_y$ and $\mathbf{b}_{other}[n^*] := \frac{2}{3}$;
- $\mathbf{b}_i := \mathbf{b}_i + \frac{1}{3N}\mathbf{b}_y$ and $\mathbf{b}_i[n^*] := 0$,
  with $i \neq y$ and $i \neq other$.

Note that for accesses whose target is not known to S_Y, the formulas illustrated in the case of no collusion among the providers in Section 3.7.2 apply.

To evaluate entropy evolution, similarly to what has been done in the previous scenario, we performed a series of simulations considering an index with $|\mathcal{N}| = 1200$ leaf nodes, and varying the frequency of known accesses to be one every $T \in \{16, 8, 4, 2, 1\}$ requests. At initialization time, the probability matrix is uniform and then the initial value of the average entropy is the maximum theoretical one (i.e., $\log_2 1200$). Figure 3.10(a) shows that the system reaches an equilibrium between the information gain acquired by observing known accesses and the destruction of information caused by swapping. Note that entropy is higher when the frequency of accesses with known target is lower. However, the minimum entropy value reached at equilibrium is high even when the provider is supposed to know all the accesses ($T = 1$ in Figure 3.10(a)). This confirms the effectiveness of the design of our distributed index structure, showing how it is able to guarantee access and pattern confidentiality.

Collusion between two providers. We now evaluate the entropy evolution when two providers (S_Y and S_G for concreteness) collude. Consider the initial uniform configuration (see Figure 3.9(c))

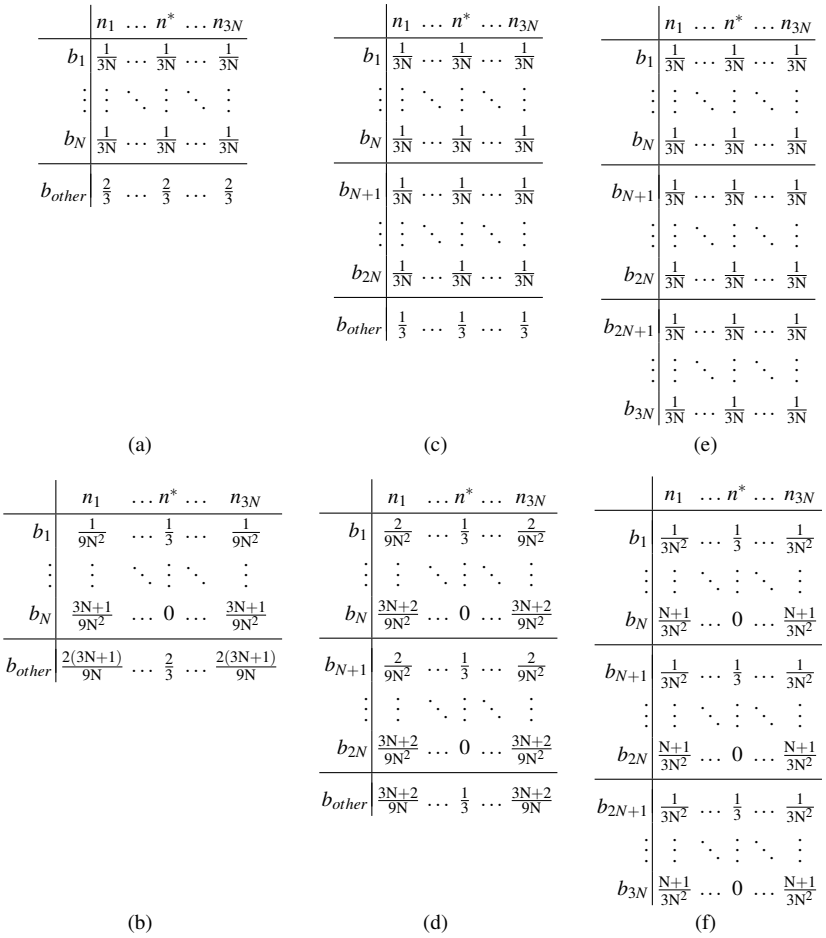


Figure 3.9: Probability matrices at initialization (a,c,e) and after the first known access (b,d,f) to blocks b_1, b_{N+1}, b_{2N+1} with: no collusion (a,b), collusion among two providers (c,d), and full collusion (e,f)

and assume that the first access has n^* as target and accesses blocks b_y at S_Y and b_g at S_G. The probability matrix evolves similarly to the case in which the providers do not collude. Because of swapping, the target n^* of the access can be allocated to any of the accessed blocks (i.e., b_y, b_g, or a block at S_B) with the same probability. Hence, $P(b_y,n^*) = P(b_g,n^*) = P(b_{other},n^*) = \frac{1}{3}$, while $P(b_i,n^*) = 0$ with b_i a block stored at S_Y or S_G different from b_y and b_g. The values of probabilities $P(b_y,n_j)$ and $P(b_g,n_j)$, with $n_j \neq n^*$, before the access are uniformly redistributed over the whole set of $3N$ blocks, including b_y, b_g, and the blocks at S_B. Assuming, as an example, that the first access is known to be for n^* and it accesses block b_1 at S_Y and block b_{N+1} at S_G, Figure 3.9(d) illustrates the probability matrix after the access execution. Extending the reasoning to a sequence of accesses, we can formalize the changes to the probability matrix due to the observation of an access with known target n^* as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g)$ and $\mathbf{b}_y[n^*] := \frac{1}{3}$;
- $\mathbf{b}_g := \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g)$ and $\mathbf{b}_g[n^*] := \frac{1}{3}$;
- $\mathbf{b}_{other} := \mathbf{b}_{other} + \frac{1}{3}(\mathbf{b}_y + \mathbf{b}_g)$ and $\mathbf{b}_{other}[n^*] := \frac{1}{3}$;

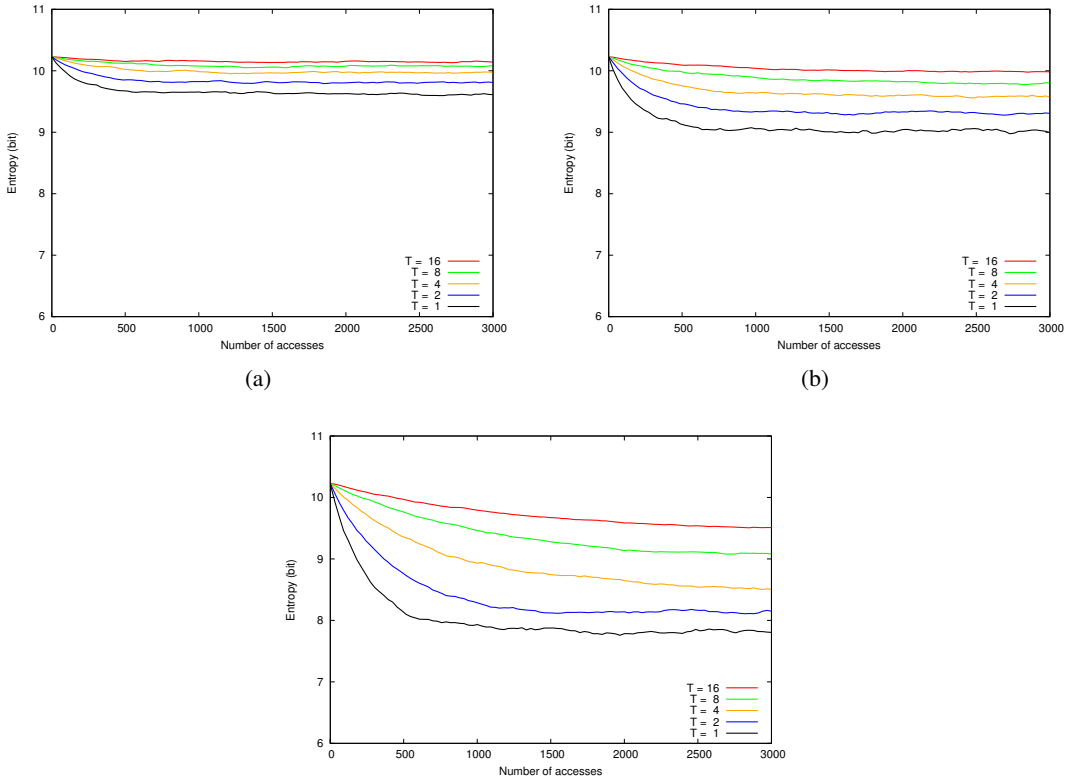


Figure 3.10: Evolution of the entropy for a distributed index with 1200 leaf nodes distributed among three storage providers starting from a no-knowledge scenario in a no collusion (a), collusion between two providers (b), and full collusion (c) scenario

- $\mathbf{b}_i := \mathbf{b}_i + \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g)$ and $\mathbf{b}_i[n^*] := 0$,
  with $i \neq y$, $i \neq g$, and $i \neq other$.

Again, for accesses whose target is not known to S_Y, the formulas illustrated in the case of collusion between two providers in Section 3.7.2 apply.

Figure 3.10(b) shows the protection (i.e., the average entropy value) offered by an index with $|\mathcal{N}| = 1200$ leaf nodes distributed among three storage providers that know the content of the node retrieved by the client one every $T \in \{16, 8, 4, 2, 1\}$ access requests. As expected, the equilibrium is reached at a lower entropy value compared with the case in which there is no collusion. However, the degree of protection exhibited by our solution still guarantees pattern confidentiality.

Full collusion. We now evaluate the knowledge gain (i.e., how entropy decreases) in presence of collusion among all the three providers. Consider an initial configuration of absence of knowledge in the node-block allocation (Figure 3.9(e)). Assume now that the first observation is for an access with n^* as target and over b_y at S_Y, b_g at S_G, and b_b at S_B. By colluding, the providers can share information on the leaf block accessed at each provider, therefore they know for sure that n^* is allocated at one among b_y, b_g, and b_b with equal probability. Hence, $P(b_y, n^*) = P(b_g, n^*) = P(b_b, n^*) = \frac{1}{3}$, while $P(b_i, n^*) = 0$ for each b_i with $i \neq y$, $i \neq g$, and $i \neq b$. Again, the values of probabilities $P(b_y, n_j)$, $P(b_g, n_j)$, and $P(b_b, n_j)$, with $n_j \neq n^*$, before the access are uniformly redistributed over the whole set of $3N$ blocks, including the accessed blocks. These changes in the

	Shuffling	Swapping
Two providers	2.5	8.8
Three providers	5.8	9.2

Table 3.1: Comparison among the rates of entropy increase (bits/10^4 accesses) for shuffling and swapping with two and three providers, assuming no collusion.

probability matrix are illustrated in Figure 3.9(f), which assumes, as an example, that the known target of the first access is n^* and that blocks b_1 at S_Y, b_{N+1} at S_G, and b_{2N+1} at S_B are accessed.

For each access request, we can formalize the changes to the probability matrix due to the observation of the access to blocks b_y, b_g, and b_b like for the case of full collusion described in Section 3.7.2, except for the T-th access whose target is known, for which the update of the probability matrix is as follows (the values in the right side of the formulas are those before the access):

- $\mathbf{b}_y := \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g + \mathbf{b}_b)$ and $\mathbf{b}_y[n^*] := \frac{1}{3}$;
- $\mathbf{b}_g := \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g + \mathbf{b}_b)$ and $\mathbf{b}_g[n^*] := \frac{1}{3}$;
- $\mathbf{b}_b := \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g + \mathbf{b}_b)$ and $\mathbf{b}_b[n^*] := \frac{1}{3}$;
- $\mathbf{b}_i := \mathbf{b}_i + \frac{1}{3N}(\mathbf{b}_y + \mathbf{b}_g + \mathbf{b}_b)$ and $\mathbf{b}_i[n^*] := 0$,
  with $i \neq y$, $i \neq g$, and $i \neq b$.

Figure 3.10(c) shows the protection (i.e., the average entropy value) offered by an index with $|\mathcal{N}| = 1200$ leaf nodes distributed among three storage providers that know the content of the node retrieved by the client one every $T \in \{16, 8, 4, 2, 1\}$ access requests. As expected, the equilibrium is reached at a lower entropy value compared with the cases of no or partial (between two providers) collusion. However, it still presents a significant level of obfuscation of the correlation between nodes and blocks, demonstrating the effectiveness of the swapping technique.

3.8 Discussion

The main characteristics of our approach are the use of swapping (in contrast to shuffling) and the use of three providers for managing the data structure. In this section we provide the motivation behind these choices. We close the section with a comment on the performance and cost of our approach.

Why swapping. Swapping forces an accessed node to be reallocated to a block at a different provider from the one where the node was stored before the access. By contrast, with random shuffling, the node can remain with the provider (i.e., under its control) with probability $\frac{1}{S}$, where S is the number of providers (three in our approach). Considering the worst case scenario illustrated in Section 3.7.2, in absence of collusion (as the system is expected to work) swapping provides a much higher increase of entropy (i.e., a much higher degradation of the knowledge) at each provider. Table 3.1 reports the rate of entropy increase after the first access starting from the full knowledge initial configuration, comparing shuffling and swapping techniques combined with the use of two and three providers (we will discuss on the number of providers next), and assuming

no collusion. As it is clear from the table, swapping outperforms shuffling, with entropy increase being three times as much in the case of two providers and twice as much in the case of three providers. The analysis in Section 3.7.3 further confirms this observation. In fact, even if entropy initially decreases, it never goes below a minimum threshold (i.e., maximum knowledge), which is higher if the providers do not collude.

Why three providers. Collusion, while unlikely, cannot be completely ruled out (or in any case some protection must be provided against it). When only two providers are used, the requirement of swapping to move the accessed node out of its original block implies a deterministic reallocation of the node (to the other provider). In case of collusion between the two providers this discloses the block to which the node is re-allocated. Assuming the worst initial configuration in Section 3.7.2, the determinism of the swapping operation would provide then no increase in entropy in presence of collusion. Figure 3.8(b) illustrates the entropy evolution for an index with 1200 leaf nodes with the use of two providers. As the figure shows while two providers provide an increase in entropy (as also reported in Table 3.1) in case of no collusion (solid black line), the entropy would show no increase in the case of collusion (dotted red line). By contrast, as already discussed in the Section 3.7.2 and visible in Figure 3.8(a), the use of three providers shows an increase of entropy (and hence protection due to the degradation of the knowledge of the providers) even when all providers collude. This is also testified by the lower decrease of the entropy obtained when the providers do not have any initial knowledge, but they know the target of each access. In fact, as visible in Figure 3.10, the minimum entropy reached when the providers do not collude is 7.3% higher than the case of two colluding providers, and 18.7% higher than the case of all the providers colluding.

Economic considerations. One may wonder how the involvement of three providers (in contrast to one or two) impacts the overall costs of the system. In this section, we provide some economic considerations on the approach. The price lists of most Cloud providers present three cost components (we take the March 2015 prices of Amazon S3 as a reference; similar pricing schemes are used by the other providers): *1)* monthly amount of stored data (US$ 30 per month per TB); *2)* number of access requests (US$ 5 per million PUT requests, and US$ 0.4 per million GET requests); and *3)* amount of data transferred out of the provider (roughly US$ 80 per TB; data sent to the provider is free of charge). The second parameter dominates the third one when requests transfer on average less than 50KB, which is the case for our index (the node size we typically used in simulations is a few KB). A simple analysis shows that, for an index distributed at three providers, the storage and access costs are comparable when the system has to manage around 10K index access requests per day over a 1TB data collection. More precisely, when the access frequency is lower, the storage costs dominate; when the access frequency is higher, it is the cost of upload (PUT) requests that dominates.[1] However, for systems with a low ratio between access frequency and storage size the costs for a solution involving three providers is comparable with those incurred for a solution with a single provider. This is due to the fact that the overall memory needed for our distributed index structure is independent of the number of providers on which the index is stored, and the access cost will linearly increase with the number of providers used.

[1] PUT requests are 12.5 times more expensive than GET requests; in our distributed index PUT and GET requests occur with similar frequency, then the cost of PUT requests dominates that of GET requests.

3.9 Summary

Building on the shuffle index approach studied in Work Package 2, we have proposed to distribute data storage among multiple Cloud providers to the aim of providing better guarantees of access and pattern confidentiality. The proposal developed within ESCUDO-CLOUD is based on the distribution of the shuffle index among three independent servers and has the advantage of proving protection guarantees also in case of collusion. Within ESCUDO-CLOUD we formally analyzed the protection offered by our approach to measure its quality, considering two representative scenarios. We considered first a worst-case scenario where providers start with a complete knowledge of the data they store, showing how swapping quickly brings to a degradation of such a knowledge. We also analyzed a scenario where the providers do not have initial knowledge, but know the individual accesses, and show how our approach prevents knowledge accumulation. Our analysis confirms that distributed allocation and swapping provide nice protection guarantees, even in presence of collusion.

4. Data Protection as a Service for Federated Cloud Storage

In this chapter we showcase the set of security tools, techniques and components that have been used to design and develop the prototype software solution for Use Case 3 (Data Protection as a Service use case) for the ESCUDO-CLOUD project. This solution will allow BT customers to protect and control their confidential and sensitive information with a user-friendly data protection service that will keep data private and help meet regulatory compliance requirements. Customers will be able to store their data on multiple Cloud vendors and platforms, and will be able to manage the security related aspects of their stored data via the federated protection service. Only the customer (or a trusted third party designated by the customers) will have the access and control of the decryption keys, giving the freedom to decrypt data on-demand and in real time.

The main problem being addressed in this use case is 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, and PCI DSS 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 Application Programming Interface (API), specialized services, and security functionalities to satisfy various user requirements. Therefore, the tools and techniques used or implemented for Use Case 3 will be able to construct a Cloud security service that provides data protection service to a customer for block and object storage services, and works seamlessly across multiple Cloud platforms.

The rest of this chapter is structured as follows:

Section 4.1 provides the details of the solution's contribution towards Use Case 3 objectives and how those objectives will be realized.

Section 4.2 presents the state of the art for data encryption services offered in the commercial Cloud platforms.

Section 4.3 highlights the innovative features contributed by the BT DPaaS solution in ESCUDO-CLOUD.

Section 4.4 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 4.5 describes the design of the current high level architecture, the granularity of the

main components, and the approach towards final solution design. It also provides the implementation details about the prototype developed to show the feasibility of the proposed solution.

Section 4.6 draws a few concluding remarks about this chapter and also provides a mapping of the tools & components to the exact functional requirements of Use Case 3. It also details the deliverable plan of the future work up to the end of the project.

4.1 Background

In this section, we try to showcase the relevance of the work done in the design and development of the DPaaS solution towards the ESCUDO-CLOUD Use Case 3. This is done by demonstrating the contribution of the security tools, techniques and components of the DPaaS solution towards the objectives of Use Case 3.

Objective 1: The first objective of the ESCUDO-CLOUD Use Case 3 is the application of data protection in Multi-Cloud environments.

Realization: This objective is realized by designing a solution that offers DPaaS via a Cloud service store that enables the customers to manage the security life-cycle of their data stored on multiple Cloud platforms, called the BT Service Store. The design and features of this component will be discussed in Sections 4.4 and 4.5. As such, the BT Service Store component is further applicable for use by Managed Security Service Providers (MSSP) and for Cloud customers and user organizations that want to control the protection of data-at-rest across multiple Cloud environments.

Objective 2: The second of the ESCUDO-CLOUD Use Case 3 objectives is to only allow the customers or trusted third parties of their choice to manage the encryption keys in accordance with policy-based access control rules.

Realization: This objective is addressed by the BT DPaaS solution by applying uniform and tightly-coupled data protection and data access policies for heterogeneous data stored in a multiplicity of Cloud providers (both private and public Cloud platforms). The data is protected by leveraging a Cloud-based data protection service for encrypting data independently and transparently of the Cloud provider hosting it, and ensuring that Cloud service providers have no access to the encryption keys or its protection and access control policies. As shown in the upcoming Sections 4.4 and 4.5, the Access Control Service and the Key Management Service components of the DPaaS are constructed and offered as tightly integrated services that manage the protection of the sensitive outsourced data. This tight integration ensures that the decryption of the protected data is only possible in the client's environment following a policy-based approval procedure and the resulting release of the encryption key.

Objective 3: The third ESCUDO-CLOUD Use Case 3 objective is to allow the customers to choose and use different Cloud storage services and provide an integrated view of these multiple Cloud storage services to the customers.

Realization: This objective is realized by the DPaaS solution by utilizing a combination of the BT Service Store component and customized data encryption agents. The BT Service Store is used to offer customers centralized access and management interfaces to different Cloud storage services like block storage, object stores and Big Data clusters that can be hosted on or provided by different Cloud service providers. To cater for the diverse architectural and operational realities of these disparate storage services, the data encryption agents are highly customized. In case of block storage services they fulfill their roles as software agents installed inside individual virtual machines, in case of object stores they fulfill their functions as part of a proxy gateway through

which the data objects transit, and in case of Big Data storage clusters they will be designed and implemented as HDFS plug-ins that can understand the Hadoop filesystem virtualisation and encrypt and decrypt data on the HDFS Data Nodes accordingly.

Objective 4: The fourth and last ESCUDO-CLOUD Use Case 3 objective is to ensure that the data is secured in accordance to specific data protection regulations and standards e.g., FIPS etc.

Realization: The data encryption agents, gateways and plug-ins of the DPaaS solution are responsible for performing the core encryption and decryption operations on the customers data. The encryption and decryption process is transparent to the applications and end-users while the data-at-rest will always stay in encrypted state on the multiple Cloud platforms, in compliance with specific data security standards and regulations. Furthermore, the specific encryption library used by the BT Data Encryption Agents to perform these low-level tasks is FIPS-140-2 compliant.

4.2 State of the Art

Commercial Cloud computing providers such as Amazon and Google have started to provide data encryption services for their data storage offerings [AMZ16, GOO16]. Their encryption services, however, are rather simple and have a number of limitations such as vendor lock-in and inflexibility in key management. In addition to these already deployed services, a number of other services were introduced as a result of various research efforts. In particular, the authors of [IKC09] introduced a Privacy-as-a-Service, which is implemented by tagging the data according to three categories of privacy (essentially low, medium and high), and then applying pre-mapped encryption algorithms on the data to ensure its security in the Cloud architecture. This service allows for the secure storage and processing of users' confidential data by leveraging the tamper-proof capabilities of cryptographic coprocessors, which requires specialized hardware devices.

On the other hand, the authors of [KVA14] designed ESPRESSO, an Data Encryption as a Service for Cloud Storage Systems. This service was designed for integration with two open-source Cloud storage platforms: Swift in OpenStack [OPS16] and Cumulus in Nimbus [NIM16]. Authors of [SG14] further discussed principles and architecture of a Secret Storage as a Service and subsequently presented a design for such service. This service provides secure secret storage and access control from the applications to securely storing, tracking, and controlling access to digital secrets such as cryptographic keys and hashed passwords. Our proposed framework is different from these services in the sense that we provide a full life-cycle of data security management i.e., creation, provisioning, operation and de-provisioning of the service for each customer. Our solution is also able to encrypt or decrypt existing data in-place, so the customers don't have to move data around just to have it secured. This contributes towards maximizing the flexibility of users in using the data encryption service. Besides, our framework is extensible to work with different Cloud platforms and security solutions.

The closest works to ours are [DDEM$^+$14] and [PSDC15]. Specifically, with respect to the work in [DDEM$^+$14], our service is similar because both services target Multi-Cloud environments and provide several options for users to customize the services according to their needs. The difference, however, is that the security service in that work focuses on protecting the VM with firewalls and security tools while our focus is specifically on data encryption. On the other hand, our work can be considered as an extension of the work presented in [PSDC15]. Both works share the same idea of providing data encryption as a service. However, while [PSDC15] simply presented primitive ideas and shown a high-level hard-coded architecture, our work introduced matured ideas with a clear, low-level design architecture. Besides, the working principles between

this work and ours are also very different, as they assumed a fully working federated infrastructure already in place, having comprehensive monitoring capabilities of services running on it.

4.3 ESCUDO-CLOUD Innovation

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.

4.4 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 Data Protection as a Service (DPaaS). As it is obvious from the title of this document, this is not an exhaustive and permanent list, but we are quite confident that the tools and technologies mentioned in this section will be sufficient to address the bulk of the requirements identified in Use Case 3 of the ESCUDO-CLOUD in a Multi-Cloud environment to enterprise customers.

The main 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 will be able to utilize these services via a service store that integrates the protection system and its components with multiple independent Cloud service providers. Therefore, the service store's orchestration capabilities is the central component of the data protection service.

4.4.1 Service Orchestrator

Purpose

The main purpose of the BT Cloud service store's Service Orchestrator is to provision the data protection service to the customers and manage its life-cycle. Each customer gets a 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 the following:

- 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 ESCUDO-CLOUD is the ability to support the data security capabilities across multiple Cloud environments. This is where the Service Orchestrator comes into use as it is designed in a manner so that the Cloud infrastructure and operating system level dependencies are decoupled. This enables it to achieve the goal of provisioning the data protection service into a wide variety of Cloud targets, both public and private. Currently it 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.

4.4.2 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 of multiple X.509 digital certificate formats like PKCS#7, PKCS#8, DER, PEM, and PKCS#12.
- Supports 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, as well as the main objective of the ESCUDO-CLOUD, is to enable for the end-users the maximum control and ownership of their data in the Cloud eco-system. This is realized in Use Case 3 by the customer-based management of the encryption 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 and the key management service have no view or control of the customers' keys and other security credentials, even if the key management service is deployed on BT Cloud platform.

4.4.3 Access Control Service

Purpose

The main purpose of the BT Access Control Service is to allow the 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 the untrusted or curious Cloud service providers. It 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, 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 the following:

- Availability of comprehensive access controls, especially for block storage and file-system based Cloud storage services.
- Ability to enforce least-privileged user, 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

As mentioned previously, one of the core requirements of Use Case 3 and a main objective of the ESCUDO-CLOUD, is to enable for the end-users the 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. In case 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.

4.4.4 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, virtualised, object, and big data storage environments. It implements a transparent encryption approach at the file system 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 the 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.

4.5 Detailed Architecture

4.5.1 Current High-level DPaaS Design

The current architecture of the DPaaS solution is designed around the core functional requirements of a multi-tenant customer scenario requiring 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. As shown in Figure 4.1, this the central component of the BT Cloud service store that is used to provision the overall DPaaS capability to the BT customers. Each customer gets a compartmentalized access to the service store and 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 KMS interface. The BT Service Store also has the ability to install and configure the encryption agents, plug-ins and gateways on virtual machines on different supported Cloud platforms.

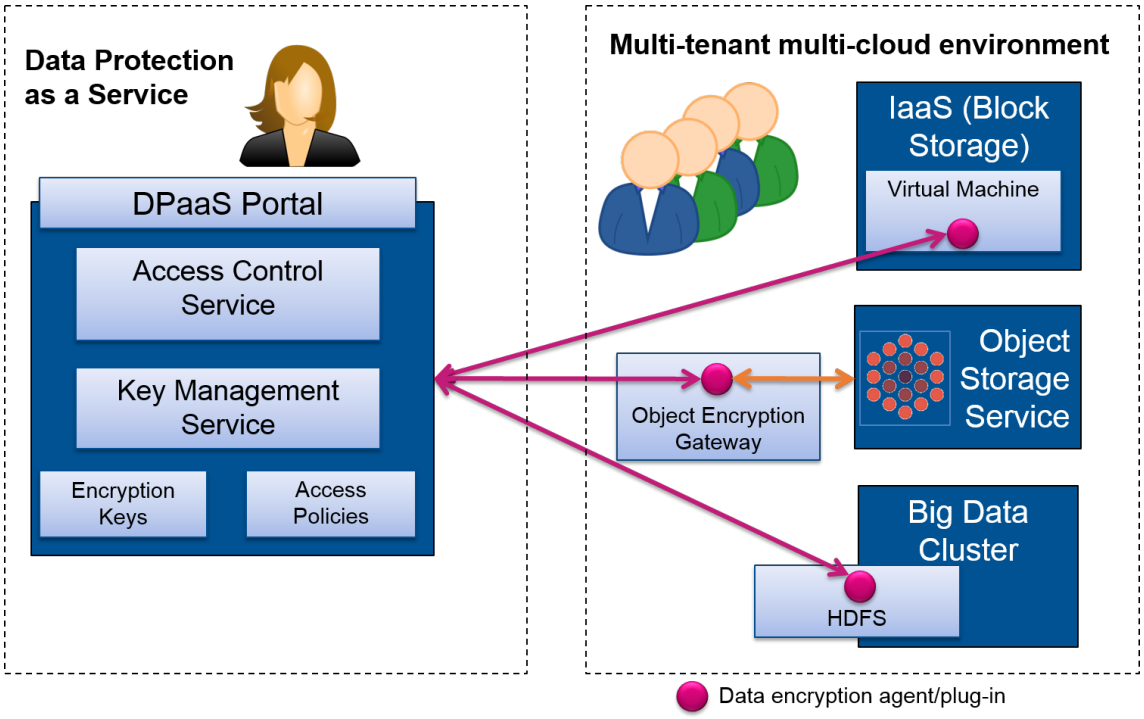


Figure 4.1: High-level View of the DPaaS Solution Design

The main benefit of this design is that a centrally managed data protection service is used to manage the functional differences of different Cloud storage services on multiple Cloud platforms. In Use Case 3, as depicted in Figure 4.1, we focus on three different storage mediums that are commonly used by BT 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, the sub-use cases will inherit a common encryption, key management and access control system but will have different abstraction layers to cater for the underlying storage mediums. The detailed architecture for two of these services has been more or less finalized and is discussed in the following sections; the Big Data storage encryption work is currently in initial phase and will be discussed briefly in the future work section 4.6.1.

4.5.2 Block Storage Encryption Design

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 the 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 VMs running on hosts in that cluster. On KVM and VMware, you can provision Primary storage on a per-zone basis. You can add multiple Primary storage servers to a cluster or zone. 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 on 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 4.2.

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

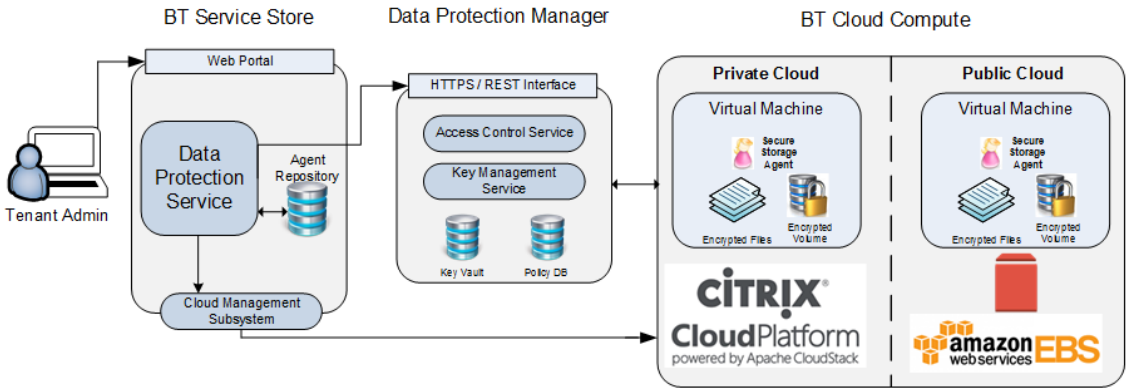


Figure 4.2: Architecture of the Block Storage Encryption Service Component of the DPaaS Solution

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 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 SSL/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 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.

4.5.3 Object Storage Encryption Design

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 Elastic Block Store

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 data's 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" which uses a variant of the Reed-Solomon error correction codes. This ensures increased up-time, 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 of one main module, i.e., the BT Cloud Encryption Gateway, and is shown in Figure 4.3.

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 keep and 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.

4.5.4 Block Storage Encryption Implementation

The detailed architecture of the block storage encryption service of DPaaS is shown in Figure 4.4. In subsequent parts of this section, we respectively explain the reference design of the prototype

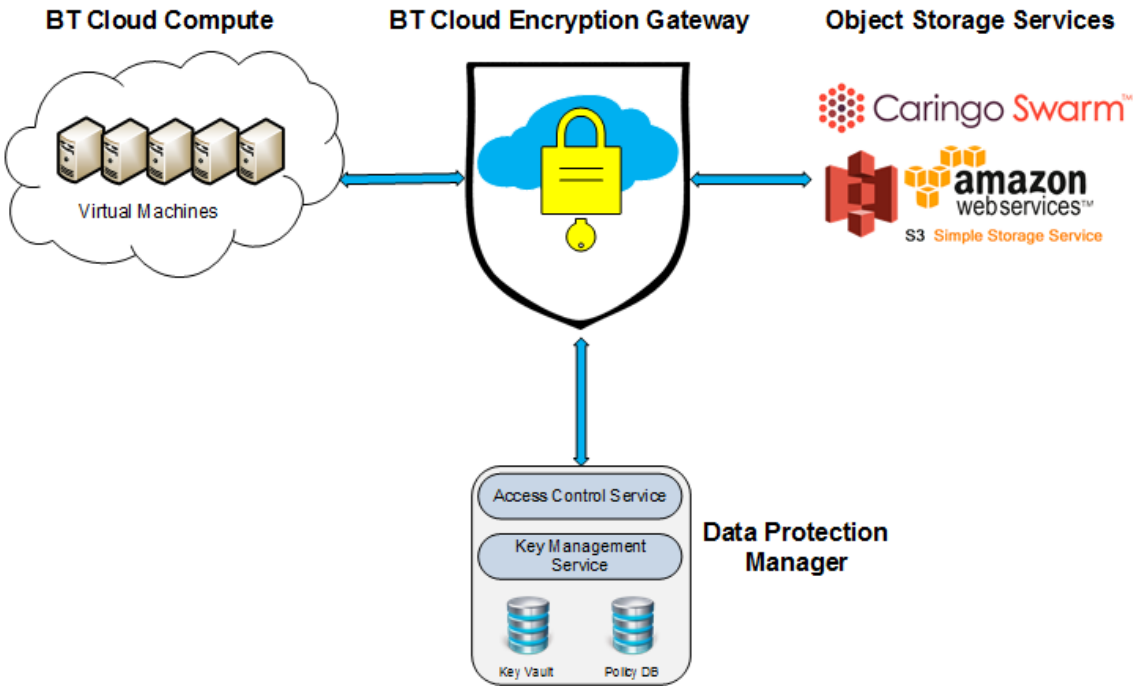


Figure 4.3: Architecture of the Object Storage Encryption Service Component of the DPaaS Solution

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.

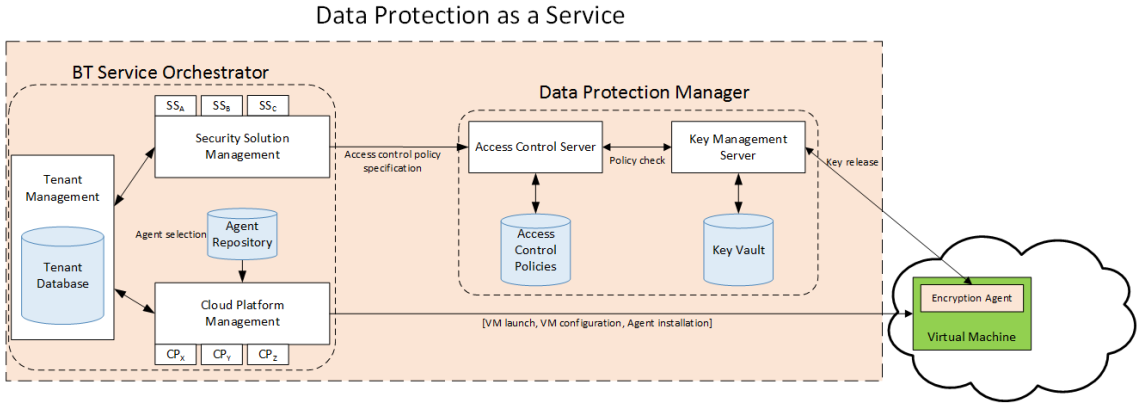


Figure 4.4: Implementation Reference of the Block Storage Encryption Service Prototype

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 VM deployment and VM 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 4.4, 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 VM 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 VM 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.

4.5.5 Object Storage Encryption Implementation

The architecture of the object storage encryption service of DPaaS is shown in detail in Figure 4.5. It reuses most of the components in 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:

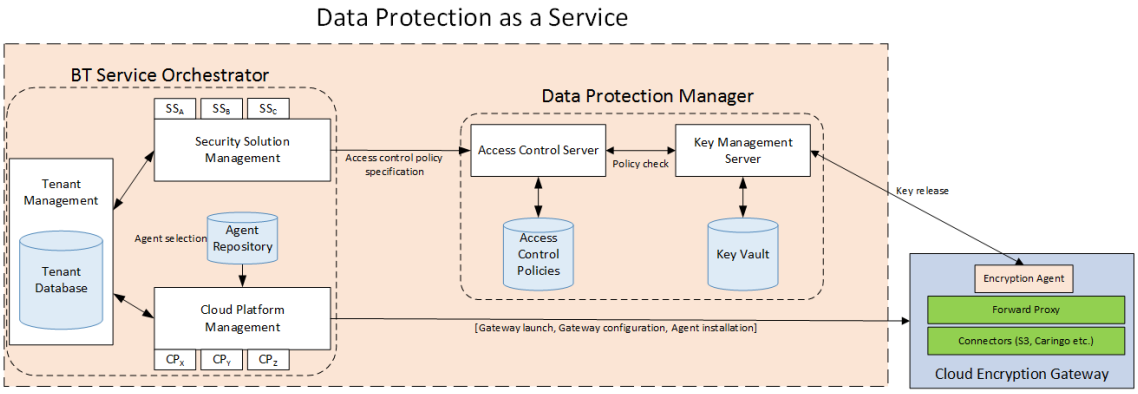


Figure 4.5: Implementation Reference of the Object Storage Encryption Service Prototype

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 Cloud Platform Management component (CPM) 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.

4.6 Summary

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.

The contribution of the security tools, techniques and components of the DPaaS solution towards the exact individual functional requirements of Use Case 3 [SDR15a, SDR15b] is given, in a summarized fashion, in Table 4.1. A more detailed description of these associations will be included in the next and final versions of this document (D4.3, D4.4).

Requirement Reference	DPaaS Component
REQ-UC3-KM-1	Data Protection Manager (Key Management Service)
REQ-UC3-KM-2	Data Protection Manager (Key Management Service)
REQ-UC3-KM-3	Data Protection Manager (Key Management Service)
REQ-UC3-KM-4	Data Protection Manager (Key Management Service)
REQ-UC3-KM-5	BT Service Store, Data Protection Manager (Key Management Service)
REQ-UC3-KM-6	BT Service Store, Data Protection Manager (Key Management Service), Data Encryption Agents
REQ-UC3-AC-1	Data Protection Manager (Access Control Service)
REQ-UC3-AC-2	Data Protection Manager (Access Control Service)
REQ-UC3-AC-3	Data Protection Manager (Access Control Service)
REQ-UC3-AC-4	Data Protection Manager (Access Control Service)
REQ-UC3-AC-5	BT Service Store, Data Protection Manager (Access Control Service)
REQ-UC3-AC-6	Data Protection Manager (Access Control Service, Key Management Service)
REQ-UC3-AC-7	Data Protection Manager (Access Control Service, Key Management Service)
REQ-UC3-SO-1	BT Service Store
REQ-UC3-SO-2	BT Service Store
REQ-UC3-SO-3	BT Service Store
REQ-UC3-SO-4	BT Service Store
REQ-UC3-SO-5	BT Service Store
REQ-UC3-SO-6	BT Service Store
REQ-UC3-SO-7	BT Service Store
REQ-UC3-SO-8	BT Service Store
REQ-UC3-DE-1	BT Service Store, Data Protection Manager (Access Control Service)
REQ-UC3-DE-2	BT Service Store, Data Encryption Agents
REQ-UC3-DE-3	Data Encryption Agents
REQ-UC3-DE-4	Data Encryption Agents, Data Protection Manager
REQ-UC3-DE-5	Data Encryption Agents, Data Protection Manager

Table 4.1: Mapping Between UC3 Functional Requirements and DPaaS Components

4.6.1 Future Work

The main work to be done in this task in the next year will revolve around incorporating another Multi-Cloud storage service in the DPaaS eco-system, i.e., Big Data storage. The specific Big Data technology that has been identified and chosen to be included within the scope of DPaaS is the Hadoop Distributed File System (HDFS). HDFS is a highly fault-tolerant distributed filesystem 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. 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 can probably re-use some of the work done in the design and development of the block storage encryption component for Big Data storage clusters.

Another area we are considering for future enhancement is the granularity of the access control attributes. In addition to encryption and key management, the DPaaS solution can be extended to enforce very granular, least-privileged user access management policies, enabling protection of data from misuse by privileged users and APT attacks. For example, granular privileged user access management policies can be applied by users, processes, file types, time of day, and other parameters. These enforcement options are very granular and they can be used to control not only permission to access clear-text data, but what also what file-system commands are available to the customers.

5. Conclusions

In Chapter 2, we have introduced a SLA-based methodology for the Multi-Cloud environment that involves: *i)* MCSLA based services construction and management and *ii)* services selection. In the former, we investigated the issues of MCSLA hierarchy and considered the dependencies between different SLOs. Moreover, our approach automatically validates the MCSLA by exploring dependencies between SLOs. In the latter, we developed a selection algorithm to score the CSP-offered services based on the given SLAs and also the degree of customer satisfaction.

In Chapter 3, we have illustrated a technique for protecting confidentiality in Multi-Cloud scenarios. The proposed approach protects both the confidentiality of data stored at external Cloud providers and the accesses to them. This approach is based on the use of a key-based dynamically allocated data structure studied in Work Package 2, the shuffle index, that is distributed over three independent Cloud providers. The security analysis confirms that distributed allocation and swapping provide nice protection guarantees, even in presence of collusion among Cloud providers. We are currently working on the implementation of a distributed shuffle index proving integration with different real-world Cloud providers, to evaluate the protection guarantees and the performance overhead of our proposal.

In Chapter 4, we cover the design and development of the DPaaS solution that is being developed 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 description has been provided 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. 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 and object 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's protection. This is achieved due to the marriage 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.

Bibliography

[AF08] D. Ameller and X. Franch. Service level agreement monitor (SALMon). In *Proc. of International Conference on Composition-Based Software Systems*, pages 224–227, 2008.

[AMZ16] Amazon Encryption Service. *http://docs.aws.amazon.com/AmazonS3/latest/dev/serv-side-encryption.html*, 2016.

[BJMU11] K. Bernsmed, M. Jaatun, P. Meland, and A. Undheim. Security SLAs for Federated Cloud Services. In *Proc. of Availability, Reliability and Security*, pages 202–209, 2011.

[BPSMS07] M. Boniface, S. Phillips, A. Sanchez-Macian, and M. Surridge. Dynamic service provisioning using GRIA SLAs. In *Proc. of Service-Oriented Computing*, pages 56–67, 2007.

[CDF+10] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM TISSEC*, 13(3):22:1–22:33, 2010.

[CMS99] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Proc. of EUROCRYPT 1999*, pages 402–414, May 1999.

[DDEM+14] J. Daniel, T. Dimitrakos, F. El-Moussa, G. Ducatel, P. Pawar, and A. Sajjad. Seamless Enablement of Intelligent Protection for Enterprise Cloud Applications through Service Store. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1021–1026, December 2014.

[DFLS15] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Selective and private access to outsourced data centers. In S.U. Khan and A.Y. Zomaya, editors, *Handbook on Data Centers*, pages 997–1027. Springer, 2015.

[DFLS16] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Practical techniques building on encryption for protecting and managing data in the cloud. In P. Ryan, D. Naccache, and J.-J. Quisquater, editors, *Festschrift for David Kahn*, pages 205–239. Springer, 2016.

[DFM+16] S. De Capitani di Vimercati, S. Foresti, R. Moretti, S. Paraboschi, G. Pelosi, and P. Samarati. A dynamic tree-based data structure for access privacy in the cloud. In *Proc. of IEEE CloudCom 2016*, December 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 2011*, pages 710–719, June 2011.

[DFP+13a] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Distributed shuffling for preserving access confidentiality. In *Proc. of ESORICS 2013*, pages 628–645, September 2013.

[DFP+13b] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Supporting concurrency and multiple indexes in private access to outsourced data. *JCS*, 21(3):425–461, 2013.

[DFP+14] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Protecting access confidentiality with data distribution and swapping. In *Proc. of BDCloud 2014*, pages 167–174, December 2014.

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

[DFP+15b] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Three-server swapping for access confidentiality. *IEEE TCC*, 2015. pre-print.

[DFP+16] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Access control for the shuffle index. In *Proc. of DBSec 2016*, pages 130–147, July 2016.

[DFS15a] S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Data protection in cloud scenarios. In *Proc. of DPM QASA 2015*, pages 3–10, September 2015.

[DFS15b] S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Data security issues in cloud scenarios. In *Proc. of ICISS 2015*, pages 3–10, December 2015.

[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.

[Far14] S. Farokhi. Towards an sla-based service allocation in multi-cloud environments. In *Proc. of Cluster, Cloud and Grid Computing*, pages 591–594, 2014.

[FHT+12] A. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. Badia, K. Djemame, et al. OPTIMIS: A holistic approach to cloud service provisioning. *In Future Generation Computer Systems*, 28(1):66–77, 2012.

[GB14] N. Grozev and R. Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. 44(3):369–390, 2014.

[GOO16] Google Cloud Storage. *http://googlecloudplatform.blogspot.co.uk/2013/08/google-cloud-storage-now-provides.html*, 2016.

[GVB13] K. Garg, S. Versteeg, and R. Buyya. A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012–1023, 2013.

[HIML02] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD 2002*, pages 216–227, June 2002.

[HQL+11] F. Hu, M. Qiu, J. Li, T. Grant, D. Tylor, S. McCaleb, L. Butler, and R. Hamner. A review on cloud computing: Design challenges in architecture and security. *CIT*, 19(1):25–55, 2011.

[IKC09] W. Itani, A. Kayssi, and A. Chehab. Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures. In *Proceedings of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 711–716, December 2009.

[IKK14] M.S. Islam, M. Kuzu, and M. Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In *Proc. of CODASPY 2014*, pages 235–246, March 2014.

[JTS12] F. Jrad, J. Tao, and A. Streit. Sla based service brokering in intercloud environments. In *Proc. of International Conference on Cloud Computing and Services Science*, pages 76–81, 2012.

[KVA14] S. Kang, B. Veeravalli, and K. M. M. Aung. ESPRESSO: An Encryption as a Service for Cloud Storage Systems. In *Proceedings of the International Conference on Autonomous Infrastructure, Management, and Security (AIMS)*, pages 15–28, June 2014.

[LC04] P. Lin and K.S. Candan. Hiding traversal of tree structured data from untrusted data stores. In *Proc. of WOSIS 2004*, pages 314–323, April 2004.

[LDK04] H. Ludwig, A. Dan, and R. Kearney. Cremona: an architecture and library for creation and monitoring of ws-agreents. In *Proc. of Service oriented computing*, pages 65–74, 2004.

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

[LO13] S. Lu and R. Ostrovsky. Distributed Oblivious RAM for secure two-party computation. In *Proc. of TCC 2013*, pages 377–396, March 2013.

[MOS16] MOSAIC EU Project. *http://www.mosaic-fp7.eu/*, 2016.

[NIM16] Nimbus Project. *http://www.nimbusproject.org*, 2016.

[NIS11] NIST. Special 9 Publication 500-291. *Cloud computing standards roadmap-version 1.0*, 2011.

[OGTU14] O. Ohrimenko, M.T. Goodrich, R. Tamassia, and E. Upfal. The Melbourne Shuffle: Improving oblivious storage in the cloud. In *Proc. of ICLAP 2014*, pages 556–567, July 2014.

[OPS16] OpenStack. *https://www.openstack.org*, 2016.

[OS07] R. Ostrovsky and W. E. Skeith, III. A survey of single-database private information retrieval: Techniques and applications. In *Proc. of PKC 2007*, pages 393–411, April 2007.

[Pet14] D. Petcu. Consuming resources and services from multiple clouds. *In Grid Computing*, 12(2):321–345, 2014.

[PFL15] S. Paraboschi, S. Foresti, and G Livraga. Report on data protection techniques. Deliverable D2.1, ESCUDO-CLOUD, December 2015.

[PSDC15] P.S. Pawar, A. Sajjad, T. Dimitrakos, and D.W. Chadwick. Security-as-a-Service in Multi-cloud and Federated Cloud Environments. In *Proceedings of the 9th IFIP International Conference on Trust Management (IFIPTM)*, pages 251–261, May 2015.

[PWF11] R. Prodan, M. Wieczorek, and H. Fard. Double auction-based scheduling of scientific applications in distributed grid and cloud environments. *In Grid Computing*, 9(4):531–548, 2011.

[PZM13] H. Pang, J. Zhang, and K. Mouratidis. Enhancing access privacy of range retrievals over $B+$-trees. *IEEE TKDE*, 25(7):1533–1547, 2013.

[Ram01] R. Ramanathan. A note on the use of the analytic hierarchy process for environmental impact assessment. *Journal of environmental management*, 63(1):27–35, 2001.

[Saa90] T. Saaty. How to make a decision: the analytic hierarchy process. *In European journal of operational research*, 48(1):9–26, 1990.

[SBKA13] S. Sotiriadis, N. Bessis, P. Kuonen, and N. Antonopoulos. The inter-cloud meta-scheduling (icms) framework. In *Proc. of Advanced Information Networking and Applications*, pages 64–73, 2013.

[SD10] P. Samarati and S. De Capitani di Vimercati. Data protection in outsourcing scenarios: Issues and directions. In *Proc. of ASIACCS 2010*, pages 1–14, April 2010.

[SD16] P. Samarati and S. De Capitani di Vimercati. Cloud security: Issues and concerns. In S. Murugesan and I. Bojanova, editors, *Encyclopedia on Cloud Computing*, pages 207–219. Wiley, 2016.

[SDR15a] A. Sajjad, T. Dimitrakos, and R. Rowlingson. D1.1 - First version of requirements from the use cases. Escudo-Cloud Deliverable, ESCUDO-CLOUD Consortium, June 2015.

[SDR15b] A. Sajjad, T. Dimitrakos, and R. Rowlingson. D1.2 - Requirements from the Use Cases. Escudo-Cloud Deliverable, ESCUDO-CLOUD Consortium, December 2015.

[SG14] A. Sayler and D. Grunwald. Custos: Increasing security with secret storage as a service. In *Proceedings of the 2014 International Conference on Timely Results in Operating Systems*, TRIOS'14, pages 10–10, Berkeley, CA, USA, 2014.

[SS13a] E. Stefanov and E. Shi. Multi-cloud oblivious storage. In *Proc. of CCS 2013*, pages 247–258, November 2013.

[SS13b] E. Stefanov and E. Shi. ObliviStore: High performance oblivious cloud storage. In *Proc. of IEEE S&P 2013*, pages 253–267, May 2013.

[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 2013*, pages 299–310, November 2013.

[The16a] The Open Certification Framework. Cloud Security Alliance. *https://cloudsecurityalliance.org/research/ocf/*, 2016.

[The16b] The Security, Trust & Assurance Registry (STAR). Cloud Security Alliance. *https://cloudsecurityalliance.org/star/*, 2016.

[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*, 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*, pages 284–291, 2014.

[WCRL12] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE TPDS*, 23(8):1467–1479, 2012.

[WS09] M. Winkler and A. Schill. Towards dependency management in service compositions. In *Proc. of e-Business*, pages 79–84, 2009.

[WSC08] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In *Proc of CCS 2008*, pages 139–148, October 2008.

[YZZQ11] K. Yang, J. Zhang, W. Zhang, and D. Qiao. A light-weight solution to preservation of access pattern privacy in un-trusted clouds. In *Proc. of ESORICS 2011*, pages 528–547, September 2011.

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