TECHNISCHE
UNIVERSITÄT
DARMSTADT

QUANTITATIVE TRUST ASSESSMENT IN THE CLOUD

VOM FACHBEREICH INFORMATIK (FB20)
AN DER TECHNISCHEN UNIVERSITÄT DARMSTADT (D17)

ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEUR (DR.-ING.)
GENEHMIGTE DISSERTATION VON:

M. SC. AHMED TAHA
AUS KAIRO, ÄGYPTEN

DATUM DER EINREICHUNG: 07.02.2018
DATUM DER MÜNDLICHEN PRÜFUNG: 25.04.2018

REFERENTEN:
PROF. NEERAJ SURI, PH.D
PROF. DR. JOACHIM POSEGGA

DARMSTADT 2018

Taha, Ahmed : Quantitative Trust Assessment in the Cloud
Darmstadt, Technische Universität Darmstadt,
Jahr der Veröffentlichung der Dissertation auf TUprints: 2018
URN: urn:nbn:de:tuda-tuprints-74488
Tag der mündlichen Prüfung: 25.04.2018

ERKLÄRUNG

Hiermit versichere ich, Ahmed Taha, die vorliegende Dissertation ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 7. Februar 2018

Ahmed Taha

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Neeraj Suri for building and leading a research group, in which each member can pursue and develop his own research interests individually. Thank you for providing us this freedom and for guiding me through the time of my PhD endeavor. I am also very grateful to Joachim Posegga for accepting to be my external reviewer, and to Stefan Katzenbeisser, Sebastian Faust, and Thomas Schneider for being on my committee. I would like to thank Jesus Luna and Ruben Trapero for their guidance and intensive discussions that we had throughout the years. I would like to thank Daniel, Habib, Hatem, Heng, Kubilay, Nicolas, Oliver, Patrick, Sabine, Salman, Stefan, Thorsten, Tsvetoslava, Ute, Yiqun, and Zhazira for being part of and contributing to our group.

I consider myself very lucky for having worked with some very talented students and I am very grateful for their input and support, especially Ahmed, Ataus, and Soha. I would also like to thank Jolanda and Spyrous for providing the opportunity to work on joint papers in the areas assessment techniques and privacy preserving computation technique. Also, thanks to Saied for the many interesting discussions during our joint proposal. Last but not least, I would like to thank my family and friends for their support during this exciting time.

ABSTRACT

Cloud computing offers a model where resources (storage, applications, etc.) are abstracted and provided "as-a-service" in a remotely accessible manner. In such a service-based environment, the Cloud provisioning relies on stipulated Service Level Agreements (SLAs). Such an agreement is a contract between the Cloud Service Provider (CSP) and the Cloud Service Customer (CSC) regarding the offered services. These SLAs specify the Cloud services requested by the customers and are required to be achieved by the CSPs. A variety of parameters for different aspects of a service can be included in the SLA, such as but not limited to: availability, performance, downtime and location of the data.

Although there are numerous claimed benefits of the Cloud to ensure confidentiality, integrity, and availability of the stored data, the number of security breaches is still on the rise. The lack of security assurance and transparency prevented customers/enterprises from trusting the CSPs, and hence not using their services. Unless the customer's security requirements are identified and documented by the CSPs, customers can not be assured that the CSPs will satisfy their requirements. Although the recent efforts on specification of security services using SLAs, also known as security SLAs or secSLAs, is a positive development, multiple technical and usability issues limit the adoption of Cloud secSLAs in practice. For example, multiple CSPs offer similar security services (e.g., "encryption key management") albeit with different capabilities and prices. The customers need to comparatively assess the offered security services in order to select the best CSP matching their security requirements. However, the presence of both explicit and implicit dependencies across security related services add further challenges for Cloud customers to: (i) specify their security requirements taking service dependencies into consideration, (ii) determine which CSP can satisfy these requirements in a qualitative way, and (iii) identify threats that can compromise their data ownership requirements of security, functionality and performance.

Although secSLA provides specifications for the security level to be provided, assurance mechanisms are required to validate the compliance of the enforced security mechanisms to the secSLA. The lack of security transparency on the security controls implemented in the Cloud and the diversity of the security specifications covered in the secSLA make validating the service to the secSLA a challenging task. Furthermore, the customer's compensation upon a violation is a manual time intensive process.

Finally, despite the benefits of enclosing security-related information in the secSLAs, CSPs are hesitant to release detailed information regarding their security posture for security and proprietary reasons. This lack of security transparency makes assessing and validating the offered security level and finding the best CSP matching the customer's security requirements a challenging task.

In this dissertation we address the aforementioned challenges. For challenges (i) and (ii), two evaluation techniques for conducting the quantitative assessment and analysis of the secSLA-based security level provided by CSPs with respect to a set of Cloud customer security requirements are presented. The proposed techniques help to improve the security requirements' specifications by introducing a flexible and simple methodology that allows customers to identify and represent their specific, imprecise and inconsistent security needs. The techniques automatically detect conflicts resulting from inconsistent

customer requirements and provide an explanation for the detected conflicts which in-turn allows customers to resolve these conflicts. To tackle challenge (iii) and uncover threats that can compromise the customer's data ownership requirements, a threat analysis process is presented which establishes the viability of identifying threats based on the CSPs' offered services and customers' requirements.

To validate the compliance of CSPs to the contracted services in the secSLA(s), a decentralized customer-based monitoring approach is proposed. The monitoring approach detects secSLA's violations and autonomously compensates customers according to the violation severity. The approach relies on the Ethereum blockchain to securely store monitoring logs and incorporate secSLAs as smart contracts. The compliance validation framework is implemented and its functionality is evaluated on Amazon EC2.

Finally, a system that enables (a) CSPs to disclose detailed information about their offered security services in an encrypted form to ensure data confidentiality, and (b) customers to assess the CSPs' offered security services and find those satisfying their security requirements is presented. The system preserves each party's privacy by leveraging an evaluation method based on secure two-party computation and searchable encryption techniques. The system is implemented and evaluated by applying it to existing standardized secSLAs. We show that the system's performance is practical for the presented use-case. The system is formally proved against a strong realistic adversarial model, using an automated cryptographic protocol verifier.

ZUSAMMENFASSUNG

Obwohl es zahlreiche Vorteile der Cloud gibt, um Vertraulichkeit, Intigrität und Verfügbarkeit der gespeicherten Daten zu gewährleisten, steigt die Anzahl an Verstößen weiterhin. Fehlende Sicherheit und Durchsichtigkeit halten Kunden/Unternehmen davon ab, Cloud Service Anbietern (CSPs) zu vertrauen und ihren Dienst zu nutzen. Solange die Sicherheitsanforderungen der Kunden von den CSPs nicht bekannt und dokumentiert sind, kann den Kunden nicht versichert werden, dass die CSPs ihre Anforderungen erfüllen können. Obwohl die jüngsten Anstrengungen die Sicherheit, mit Hilfe von Service Level Agreements (SLAs), auch bekannt als Sicherheit SLAs oder secSLAS , zu präzisieren eine positive Entwicklung darstellen, begrenzen viele technische und benutzerfreundliche Punkte die Annahme von Cloud secSLAs in der Praxis. Obwohl secSLA Spezifizierungen für das Sicherheitsniveau zur Verfügung stellt, sind "assurance mechanisms" erforderlich, um die Einhaltung der durchgesetzten Sicherheitsmechanismen der secSLA zu validieren. Mangel von Transparenz der Sicherheit der implementieren Sicherheitskontrollen der Cloud und die Vielfalt der Sicherheitsspezifikationen, die in secSLA enthalten sind, macht die Überprüfung der secSLA Dienste zu einer herausfordernden Aufgabe. Weiterhin ist die Kundenentschädigung bei Verletzung (des Vertrages) ein manuell zeitintensiver Vorgang. Diese Dissertation beschäftigt sich mit den oben genannten Herausforderungen. Zwei Formen der Evaluierung für die quantitative Bewertung und Analyse des auf secSLA basierten Sicherheitslevels, bereitgestellt von CSPs mit Bezug auf eine Reihe von Sicherungsanforderungen von Cloud Kunden, sind vorhanden. Die Techniken erkennen Konflikte, die aus inkonsistenten Kundenanforderungen resultieren und stellen eine Erklärung für diese zur Verfügung, was wiederum den Kunden erlaubt, diese Konflikte zu lösen. Um die Übereinstimmung der CSPs mit den Vertragsservice der secSLA(s) zu validieren, wird eine dezentrale kundenbasierte "monitoring approach" vorgeschlagen. Das "monitoring approach" erkennt secSLA Verletzungen und entschädigt automatisch Kunden gemäß der Schwere der Verletzung. Die Annäherung verlässt sich auf "Ethereum blockchain". Die Annäherung der Überwachung wird durchgeführt und seine Funktionalität wird auf Amazons EC2 bewertet.

The following publications are included in this dissertation.

[AWT+17] Soha Alboghdady, Stefan Winter, Ahmed Taha, et al. "C'Mon: Monitoring the Compliance of Cloud Services to Contracted Properties." In: *Proc. of ARES*. 2017, p. 36.

[LTTS17] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. "Quantitative Reasoning about Cloud Security Using Service Level Agreements." In: *IEEE Transactions on Cloud Computing* 5.3 (2017), pp. 457–471.

[TBL+18] Ahmed Taha, Spyros Boukoros, Jesus Luna, et al. "QRES: Quantitative Reasoning on Encrypted Security SLAs." In: *Privacy Enhancing Technologies* (2018). [Submitted].

[TMT+16] Ahmed Taha, Patrick Metzler, Ruben Trapero, et al. "Identifying and Utilizing Dependencies Across Cloud Security Services." In: *Proc. of AsiaCCS*. 2016, pp. 329–340.

[TTLS14] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. "Ahp-based quantitative approach for assessing and comparing cloud security." In: *Proc. of TrustCom*. 2014, pp. 284–291.

[TTLS17] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. "A Framework for Ranking Cloud Security Services." In: *Proc. of IEEE Services Computing*. 2017, pp. 322–329.

The following publications are related to different aspects covered in this dissertation, but have not been included.

[MTS16] Salman Manzoor, Ahmed Taha, and Neeraj Suri. "Trust Validation of Cloud IaaS: A Customer-centric Approach." In: *Proc. of TrustCom*. 2016, pp. 97–104.

[MTT+16] Jolanda Modic, Ruben Trapero, Ahmed Taha, et al. "Novel efficient techniques for real-time cloud security assessment." In: *Computers & Security* 62 (2016), pp. 1–18.

[TMS+17] Ruben Trapero, Jolanda Modic, Miha Stopar, et al. "A novel approach to manage cloud security SLA incidents." In: *Future Generation Computer Systems* 72 (2017), pp. 193–205.

[TMS17] Ahmed Taha, Salman Manzoor, and Neeraj Suri. "SLA-Based Service Selection for Multi-Cloud Environments." In: *Proc. of Edge Computing*. 2017, pp. 65–72.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

CSC	Cloud Service Customer
CSP	Cloud Service Provider
SecSLA	Security Service Level Agreement
SLA	Service Level Agreement
SLO	Service Level Objective
PV	Priority Vector
NW	Normalized Weight
CM	Comparison Matrix
AHP	Analytic Hierarchy Process
DSM	Design Structure Matrix
TFN	Triangle Fuzzy Number
MCDM	Multiple Criteria Decision Making
QHP	Quantitative Hierarchy Process
ENISA	European Union Agency for Network and Information Security
CSA	Cloud Security Alliance
CSA STAR	CSA Security, Trust and Assurance Registry
CSA CAIQ	CSA Consensus Assessments Initiative Questionnaire
CSA OCF	CSA Open Certification Framework
ISO	International Organization for Standardization
RBTA	Requirement Based Threat Analysis
QRES	Quantitative Reasoning on Encrypted Security SLAs
QeSe	Query Search Protocol

ABBREVIATIONS

S	Set of services a secSLA consists of such that $S = s_1, \ldots, s_n$.
K	Set of SLOs
k	An SLO where $k \in K$
V	The set of all values of each k in K
$v_{i,k}$	k value provided by CSP_i

$v_{CSC,k}$	CSC required value for k
$\tilde{v}_i$	TFN value k provided by CSP_i such that $\tilde{v}_i$ is represented as (l_i, m_i, u_i)
B	Broker
m	Plaintext
c	Ciphertext
k	Symmetric encryption/decryption key
pk_X	Public key of party X
sk_X	Private key of party X
$cert_X$	X.509 certificate of party X
k_X^{auth}	Authentication secret generated and sent to party X
$N_X \in \{0,1\}^\beta$	Random number generated by party X from the set of all binary strings of length β
$y \leftarrow\!\!\$\ A(x)$	On input x, algorithm A binds the output to variable y
$m_1 \| m_2$	Concatenation of m_1 and m_2
KGen	key generation
Enc	Encryption algorithm
Dec	Decryption algorithm
$Enc(pk, m)$	Takes as input a public key pk, a plaintext m, and outputs a ciphertext c
$Dec(sk, c)$	Takes as input a secret key sk, a ciphertext c, and outputs a plaintext m
$Sig(sk, m)$	Signs m with a secret key sk, and outputs a signed message s_m
$Verify(pk, s_m)$	Verifies the signature using the public key pk and returns 1 if the signature is valid for the given message and 0 if not
$H(x)$	Hash function which takes as input x, and responds with a random outputs y. Ideally, a hash function is defined as a random oracle that responds with a random output $y \in \mathcal{Y}$ to each given input for $x \in X$. Where X is the set of possible messages, $\mathcal{Y}$ is a finite set of possible digests
$t^1, \ldots, t^n$	The SLA XML file can be split into n substrings.
$t^1_{CSP_1}, \ldots, t^n_{CSP_1}$	Substrings generated by CSP_1 and are named tokens, where n is the number of tokens
$w^1_{CSC}, \ldots, w^n_{CSC}$	Substrings generated by the customer CSC and are named keywords, where n is the number of keywords

1 INTRODUCTION

Cloud computing drives the vast spectrum of both current and emerging applications, products, and services, and is also a key technology enabler for the future Internet. In such a service-based environment, service provisioning relies on a Service Level Agreement (SLA) which represents a formal contract established between the Cloud Service Customer (CSC) and the Cloud Service Provider (CSP). The SLA specifies how provisioning takes place as well as the respective rights and duties of the customer as well as the CSP. Furthermore, the SLA includes the list of Service Level Objectives (SLOs) which are the measurable elements of an SLA that specify the Cloud services' levels requested by the customers, and required to be achieved by the CSP.

Although Cloud computing direct economic value is unambiguously substantial, taking full advantage of Cloud computing requires considerable acceptance of off-the-shelf services. Specifically, both security assurance and transparency remain as two of the main requirements to enable customer's trust in CSPs. The lack of assurance and transparency, along with the current paucity of techniques to quantify security, often results in Cloud customers being unable to assess the security of the CSP(s) they are paying for. Despite the advocated economic and performance related advantages of the Cloud, two issues arise: (i) how can a (non-security expert) customer meaningfully assess if the CSP fulfills his/her security requirements?, and (ii) how can a CSP provide security assurance to customer organizations during the full Cloud service life-cycle?

In this context, a number of Cloud community stakeholders (e.g., ISO 27002 [Sta13], the European Union Agency for Network and Information Security (ENISA) [Nat13], and Cloud Security Alliance (CSA) [Clo17a]) are pushing towards the inclusion of security parameters and CSP's security implementation in Service Level Agreements (named security SLAs or secSLAs [LGLS12]). Examples of security-related information are ciphers used to encrypt data, vulnerability management/assessment procedures, minimum/average incident response times, security controls' and configuration elements such as metrics for measuring cybersecurity performance, etc.

Consequently, secSLA can be a useful mean not only to disclose information about each CSP offered security services, but also to define the services' key measurable parameters. For example, how a CSP recovers from incidents is typically defined using two measurable parameters which are the incident severity and the time to recovery [HD12]. Customers can employ these measurable parameters to continuously assess and monitor (even with a realistic level of automation) the CSP's implementation of the acquired service level. Using these assessments, customers aquire better information to compare different service offerings provided by various CSPs, and thus, can select the CSP that satisfies their requirements.

Despite the benefits of disclosing the security related information in secSLAs, Cloud customers still face major problems that need to be addressed. In the next section, we define these problems using four research questions. Furthermore, we highlight our scientific contributions and specify the references where each contribution has been published.

1.1 RESEARCH QUESTIONS AND SCIENTIFIC CONTRIBUTIONS

Research Question 1: How can customers (a) specify their security requirements without introducing conflicts resulting from the dependency relations across services, and (b) evaluate and assess the security level of CSPs according to the their precise/imprecise requirements?

Dependency relations across 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. Mainly, these dependency relations increase the difficulties for the customers in finding a single CSP that satisfies their requirements, since these relations can easily introduce conflicts. For instance, a customer may require an unachievable level of a dependent security service which causes these requirements to be impossible to satisfy. Whereas, the unachievable service level exists when a service depends on resources which are not provided by the corresponding dependent service. Moreover, a customer requirement may influence or be influenced by other requirements. Consequently, a CSP being unaware of related dependencies can erroneously agree on providing an unachievable level of security service according to the customer requirement. Consequently, the CSP will not be able to fulfill this requirement, which results in a secSLA violation.

For example, the "encryption key management" service in the Cloud depends on several factors such as: (a) the techniques used to store the encryption keys, (b) the processes specifying how keys are accessed, (c) the possibility of the key recovery, and finally (d) 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. Note that most factors also depend on each other.

As the number of Cloud security services grows, the number of dependencies across the security services also increases, making it more likely for customers to introduce conflicts. Also, it becomes harder to manually detect and identify the causes of these conflicts, especially when multiple types of dependency relations are involved. Therefore, customers need to first consider the dependencies between security services that span across their security requirements' specifications, and then assess the security services offered by different CSPs to rank different CSPs based on the customer's security requirements and priorities.

Consequently, customers need techniques to quantitatively compare the security offerings across the different CSPs according. However, most of the presented evaluation techniques [LGLS12; GVB13] require customers to: (a) provide detailed values specifying their requirements and (b) submit static weights to model their priorities. In fact, both of the aforementioned requirements depend upon expert knowledge and are time consuming. In addition, the lack of Cloud customer awareness about detailed security requirements hinders them from specifying security requirements in accurate values.

To allow customers to smoothly adopt Cloud services, it is desirable to let them express their "general" requirements or imprecise preferences in natural language phrases. By using approximate linguistic descriptors, customers are able to define their requirement exactly as desired. Therefore, it is becoming an important issue for the customers to be able to make decisions regarding how to assess and rank CSPs with respect to their secSLAs and according to the customers' precise/imprecise requirements.

Scientific Contribution 1: Proposing a secSLA based dependency representation model and a quantitative reasoning framework that allow customers to specify their requirements using precise values, linguistic descriptors and natural language phrases.

To assess the CSPs according to the customer security requirements, an analysis of the security service dependencies with the handling of all conflicts that occur due to different dependency relations across services is presented. We develop a dependency model which ascertains direct and indirect dependencies across services to highlight conflicts preventing the CSPs from satisfying customer requirements.

Furthermore, two secSLA assessment techniques are proposed which are based on Analytic Hierarchy Process (AHP) [Saa90] and fuzzy-AHP [Cha96]. The proposed techniques allow customers to:

- Specify their requirements using natural language phrases and linguistic descriptors as well as precise values. Thus, enabling both novice and expert customers to provide their security requirements according to their expertise
- Assess, compare and rank various security services provided by different CSPs. We employ membership functions to capture customers' subjective requirements and then use a hierarchical fuzzy inference system to derive precise security levels for these requirements.

The proposed dependency analysis and validation framework has been presented at AsiaCCS 2016 [TMT+16]. The proposed assessment techniques have been presented at TrustCom 2014 [TTLS14], SCC 2017 [TTLS17], and TCC 2017 [LTTS17]. The first contribution is explained in Chapter 2.

Research Question 2: How can customers identify threats that can potentially violate their data ownership requirements?

Cloud infrastructures are subject to multiple threats, which may target the technological foundation of the Cloud infrastructure (e.g., virtualized environment) or architectural aspects of the next generation of Cloud storage system (e.g., availability). Virtualization is the enabling technology for the different Cloud computing models and is expected to offer secure logical isolation between virtual machines. Recently, researchers have demonstrated that the isolation property may be violated by extracting the private key of co-located users or by exchanging secrets between colluding users [PBSL13]. Different factors affect the exploitability of these threats, such as the noise level of side-channels, or deployed mechanisms. Similarly, the perceived lack of trust and control over outsourced data raises major concerns about its security and privacy. To alleviate these concerns, a systematic approach is required in order to estimate the feasibility of exploitation of these threats in a specific operational context and under the stipulated assumptions. Additionally, these threats need to be analyzed with regards to the requirements of the customer in terms of functionality, performance and security, with the aim to properly identifying which data to be outsourced is critical so that threat-modeling activities can be prioritized.

Scientific Contribution 2: Developing a requirement based threat analysis for estimating the risks based on requirements analysis.

We developed a requirements-driven methodological process that ascertains direct and indirect dependencies across the security requirements to highlight inconsistent specifica-

tions or assumptions that can lead to security breaches. Each Cloud service is presented as a tree (Boolean or X-tree as relevant) that outlines the hierarchy of requirements providing the Cloud service. As multiple services are elaborated, the methodology explores the linkages and dependencies across all the service requirements to ascertain either their compatibility or incompatibility - the latter identifying the vulnerabilities. This linked services model is a new contribution to the area of threat modeling and has resulted in a systematic 3-step methodology (initial design of the dependency model, validation of the model, use-case requirements reorganization). The presented threat analysis scheme is explained in Chapter 3.

Research Question 3: How can customers ensure that the required security level is enforced by the CSP?

In theory, by signing the secSLA, a CSP commits to providing the specified security level throughout the full life-cycle of the service. However, in practice, the service may not comply with the security level contracted in the secSLA during its full operation time. Accordingly, the compensations to be paid to the customer upon any detected violations are specified in the secSLA. However, most of the existing security compliance approaches are proposed to help the CSP to manage security compliance of their services, whereas none of the approaches addresses compliance management from the customer side.

Hence, how can customers (i) verify if the contracted security level is actually delivered, and (ii) detect and prove any violations to the contracted service levels in the secSLA at any time during the service life-cycle? At the same time, how to prevent customers from misreporting for financial gain? Finally, how can customers get compensated autonomously in case of security breaches? The current compensation process involves: (a) the customer opening a case with the support team, (b) waiting for someone to analyze the validity of the case, and finally (c) a payment or a refund is initiated in terms of credit for future usage.

Scientific Contribution 3: Proposing an approach to continuously monitor the compliance of Cloud services to secSLAs that autonomously compensates the customer if a secSLA violation is detected.

We present a framework for secSLA compliance validation which enables Cloud customers to validate the compliance of the provided Cloud service to the contracted security level in a secSLA. The framework relies on a proposed decentralized runtime monitoring approach which builds upon the Ethereum blockchain infrastructure.

Firstly, we analyze and select the SLOs that can be measured from the customer side. The analyzed SLOs are proposed by different frameworks and standards for security controls. Secondly, measurement procedures to determine the measurable SLO values are defined. Next, a decentralized customer side monitoring scheme for validating the compliance of the service is established. Further, enforcement of the secSLA using a smart contract deployed over the Ethereum blockchain. Finally, implementing an autonomous compensation process for customers if a violation is detected.

We evaluate the functionality and performance of the monitoring scheme on Amazon EC2 [Ama17]. The proposed monitoring approach has been presented at QRES 2017 [AWT+17] and is specified in Chapter 4.

Research Question 4: Is the inclusion of security implementations information in the secSLAs risky?, and will CSPs disclose detailed security related information in their secSLAs or only general and abstract information about their security posture?

Despite the benefits of detailing the security-related information, CSPs still do not disclose this kind of information in their secSLAs for security and/or commercial reasons [HD12]. The risks of including security-related information in the secSLA where pointed out by ENISA [HD12]:

- Publicly disclosing security parameters may assist attackers in penetrating the system using a hole in the publicized data. Accordingly, the rate of malicious security breaches increases, which can be much harder to detect. This can also lead to a significant financial loss as a result of the customers' compensations.
- Publicly detailing commercially sensitive information (i.e., financial terms, service levels, cost information, or/and vulnerability descriptions which may include proprietary information, etc.) can be used by other competitors to improve their services.

Scientific Contribution 4: Proposing a formally proved system which enables CSPs to publicly disclose detailed information about their offered services in encrypted secSLAs while allowing customers to assess the CSPs' offered security services and subsequently find those satisfying their requirements.

We design and implement a system called QRES (Quantitative Reasoning on Encrypted SLAs). Our system simultaneously allows:

- CSPs to specify their services along with the key measurable parameters in secSLAs, without revealing information about the offered security parameters or commercially sensitive information.
- Customers to assess and evaluate the CSP's offered security services and to choose the best CSP matching their needs.

QRES is built around a two-party privacy preserving query over encrypted data scheme (named *QeSe*). *QeSe* allows customers to search for their requirements over encrypted secSLA blindly without neither the CSPs learning these requirements, nor the customers knowing the CPSs' secret keys.

The proposed framework has been submitted to PETS 2018 [TBL+18] and is explained in Chapter 5.

1.2 DISSERTATION OUTLINE

The outline of the dissertation is structured according to the depiction in Figure 1.1, which shows the relationship between the chapters and the scientific contributions discussed. The remainder of this dissertation is organized as follows:

- Chapter 2 presents two different techniques to conduct security assessment based on the quantitative and qualitative analysis of security information. Both techniques were empirically validated through case studies using real-world CSP data obtained from the Cloud Security Alliance. Furthermore, we introduce a novel dependency

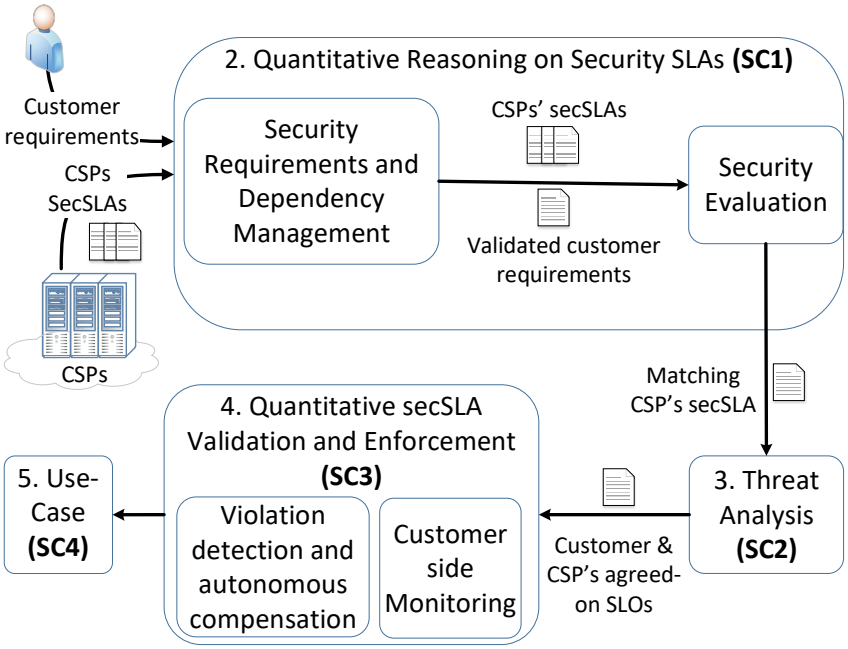

Figure 1.1.: Structure of the dissertation and scientific contributions (SC).

model and show its capability for validating each customer and CSP requirement by checking the existence of conflicts that occur due to to different dependency relations between service/SLOs.

- Chapter 3 develop a dependency model and highlights its capability for identifying threats by ascertaining and visualizing the dependencies across services. By assessing the degree of dependencies across the services, the potential threats are identified and the critical areas where services should be protected are specified.
- Chapter 4 presents the compliance validation framework. The framework validates the compliance of the service throughout the Cloud's full life-cycle by continuously testing the SLO values.
- Chapter 5 presents a novel system which enables CSPs to create secSLAs and publish them encrypted. At the same time, customers can find the CSP matching their security needs by contrasting their requirements against the encrypted secSLAs.
- Chapter 6 summarizes the contributions of the dissertation.

Part I

QUANTITATIVE REASONING ON SECURITY SLAS

2

QUANTITATIVE FRAMEWORK FOR ASSESSING CLOUD SECURITY

The quantitative security level assessment of CSPs is the primary objective of the proposed framework described in this chapter. Using this assessment approach, the CSPs are ranked (as per their secSLA's) as to how well they match Cloud Service Customer (CSC) requirements.

To achieve this, we first analyze the customer requirements according to the customer security expertise. Second, we create a dependency model to capture information about secSLAs' services and the dependencies that occur between them. Using this model, both the CSP's offered services and customers requirements are validated by checking service conflicts and different SLO's compatibility issues. Next, the validated services are structured in a precise order to easily assess these services and rank the CSPs according to customer requirements. We propose two different evaluation and assessment techniques to assess and rank the CSPs according to the customer requirements. Finally, we validate the framework by applying it to real-world data that leverages the standardized Cloud SLA's structure proposed in the ISO/IEC 19086 standard [Int14].

2.1 MOTIVATION AND CONTRIBUTION

In the Cloud area, security parameters and mechanisms are typically specified in the form of secSLAs to model security among both CSPs and customers. As the primary interfaces between the CSP and the customer are these secSLAs (which are primarily textual legal contracts of a mixed qualitative and quantitative nature), the need exists to be able to: (a) quantitatively specify and assess customer/CSP security requirements, (b) consider the dependencies between security services that span across customer/CSP security specifications, and (c) compare the security services offered by the different CSPs based on the customer's both certain and uncertain security requirements.

Although the state-of-the-art predominantly focuses on the methodologies to build and represent Cloud secSLAs, there is still a conspicuous lack of techniques that: (a) quantitatively evaluate Cloud secSLAs to provide security assurance and (b) validate the secSLAs by checking the existence of conflicts that occur due to different dependency relations between services. This lack of techniques that quantitatively evaluate and validate Cloud secSLAs to provide security assurance along with the existence of multiple CSPs offering similar security services, often results in Cloud customers being unable to trust and assess the security of the CSP's provided services they are paying for.

Contributions. To tackle the aforementioned problems, we make the following contributions:

1. Developing two evaluation techniques (QHP [TTLS14] and fuzzy-QHP [TTLS17]) for conducting the quantitative assessment and analysis of the secSLA-based security level provided by CSPs' with respect to a set of Cloud customer security requirements. These proposed techniques improve the specification of security requirements by introducing a flexible and simple methodology that allows customers to identify and represent their "specific" and "vague/uncertain" security needs. To

capture customers' subjective requirements, we deploy membership functions and then use a hierarchical fuzzy inference system to derive precise security levels for these requirements.

2. Proposing a dependency representation model for validating the secSLAs by checking the existence of conflicts that occur due to different dependency relations between services. The process of analyzing conflicts is both iterative and interactive. Furthermore, representing the security requirements in an easy dependent ordered structure using Design Structure Matrix (DSM) [Ste81]. In this structure, the security services are ordered according to their level of dependency. This makes the secSLA services and dependencies explicit and traceable regardless the number of security services. This structure assists customers who could not resolve the conflicts and thus could not specify their requirements.

3. The developed techniques along with the developed dependency model are validated using four use-case scenarios, leveraging actual real-world CSP SLA data derived from the publicly available Cloud Security Alliance's Security, Trust and Assurance Registry [Clo17d], which is compliant with the relevant ISO/IEC 19086 standard [Int14].

2.2 BACKGROUND

In this section we provide background information, which is necessary for better comprehension of the other sections. Practically, we define the concepts of security Service Level Agreement (secSLA) which are used in all the dissertation chapters. Furthermore, we give a short introduction on the dependency relations pertinent to the Cloud secSLA, on which our proposed assessment and validation approaches in Chapters 2 and 3 are based.

2.2.1 *Security Service Level Agreements*

A Cloud secSLA describes the provided security services, and represents the binding commitment between a CSP and a customer. Basically, this outlines the desired security services, each of which contains a list of SLOs. Each SLO is composed of one or more metric values that help in the measurement of the Cloud SLOs by defining parameters and measurement rules that facilitate assessment and decision making. Based on the analysis of the state of practice presented in [LTTS17], Cloud secSLAs are graphically modeled using a hierarchical structure, as shown in Figure 2.1. The root of the structure defines the main container for the secSLA. The intermediate levels (second and third levels in Figure 2.1) are the services which form the main link to the security framework used by the CSP. The lowest level (SLO level) represents the actual SLOs committed by the CSP which are consequently offered to the Cloud customer. These SLOs are the threshold values which are specified in terms of security metrics.

It is worth noting that the process of modeling values to a quantitative metric is not straightforward, as SLOs can have varied types/ranges of composite qualitative and quantitative values. Hence, we introduce the notion of a "security level" associated to each SLO of the secSLA. To formalize this concept, we introduce the following definition:

Definition 1 *A secSLA consists of a set of services* $S = s_1, ..., s_n$. *Each service* s *consists of finite positive number* n *of SLOs* k_i*; where* $i = 1 \dots n$. *Each SLO* k_i *consists of* m *different metric*

values v_i; such that $k_i = v_{i,1}, v_{i,2}, ..., v_{i,m}$. Each value implies a different security level offered by the CSP and required by the customer. The total order of security levels k_i is defined using an order relation " $<_i$ "; such that $k_i = v_{i,1} < v_{i,2} < ... < v_{i,m}$. 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 values (m) such that $k_i = \frac{1}{m} < \frac{2}{m} < \ldots < \frac{m}{m}$.

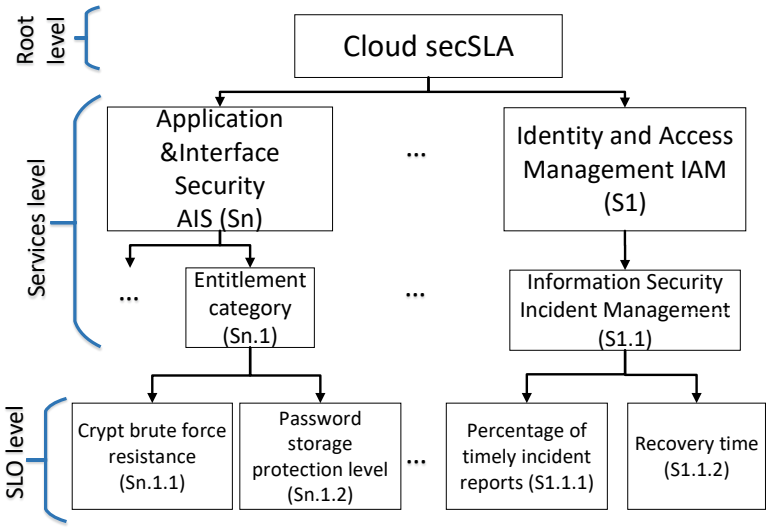

Figure 2.1.: Cloud secSLA hierarchy based on security posture provided by the STAR repository.

An example of an SLO, as shown in Figure 2.1, is "Percentage of timely incident reports" SLO which is composed of $\{yearly < half-yearly < monthly < weekly\}$ values which are defined using security levels as $level_1 < level_2 < \ldots < level_4$ respectively. These security levels correspond to $\{\frac{1}{4} < \frac{2}{4} < \ldots < \frac{4}{4}\}$. Let us consider a CSP committing "Percentage of timely incident reports" such that the CSP's secSLA specify: Percentage of timely incident report= $level_2$ such that $v_i = \frac{2}{4}$. A CSP commits other SLOs in a similar manner such that the overall CSP's secSLA contains a list of SLOs with different values that the CSP is committed to fulfill. If any of these committed values are not fulfilled by the CSP, then the secSLA is violated.

We illustrate the secSLA's structure and functionality better with an example. We consider a customer processing financial transactions using Software-as-a-Service (SaaS). The customer looks for a secSLA which specifies a recovery time objective of less than 1 minute and a monthly report offered by the selected CSP specifying the mean recovery times [HD12]. Using this example, we specify two SLOs ("Percentage of timely incident reports $S_{1.1.1}$" and "Recovery time $S_{1.1.2}$") as shown in Figure 2.1. The "Percentage of timely incident reports" SLO is composed of $\{yearly, half-yearly, monthly, weekly\}$ values which are defined using service levels as $level_1, level_2, \ldots, level_4$ respectively. If a CSP is committing a "Percentage of timely incident reports" of $monthly$, then $v_{S_{1.1.1}} = level_3$.

2.2.2 *Service Dependencies*

A service dependency is a directed relation between the services offered in Cloud scenarios. It is expressed as a $1 : n$ relationship where one service (termed as dependent)

depends on one or multiple services (termed as antecedent). 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. Explicit knowledge about dependencies is needed to support the management of secSLA by both CSPs and customers. Several types of dependencies are used in literature such as Quality of Service (QoS), price, resource and time dependencies [WSS10]. We only consider resource dependencies which are validated by matching the security values of the dependent and antecedent SLOs specified in their secSLAs.

For example, let us consider "Authentication & Authorization" Control (which defines the available authentication and user-credentials protection mechanisms) [Csi] that is composed of three different SLOs, namely "CSP authentication"[1], "User authentication and identity assurance level"[2] and "Third-party authentication support" [3]. If we consider that a CSP offering an authentication mechanism is performed by third-party (specified using "Third party authentication support" SLO in the secSLA), then the CSP does not need to specify the "User authentication and identity assurance level", as the other authentication SLOs become less relevant as authentication is performed by a third-party. This means the "Third party authentication support" SLO complements the other authentication SLOs.

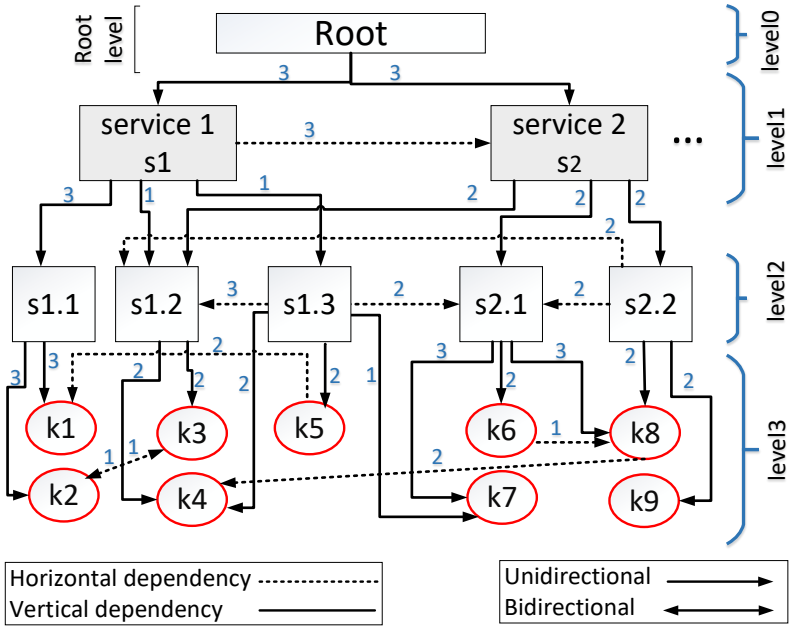

Figure 2.2.: SecSLA hierarchy showing dependencies

We classify dependencies based on their occurrence between services and/or SLOs at the same hierarchical level (horizontal dependencies), as well as between different levels in the hierarchical structure (vertical dependencies) [WS09] as shown in Figure 2.2. Dependencies can be further classified into direct and indirect dependencies. Direct dependencies occur between two interacting services. Indirect dependencies occur between services which do not directly interact with each other, but where a transitive relationship exists via an intermediate service. Figure 2.2 illustrates different dependencies such as, service $s_{2.2}$ depends on service $s_{2.1}$ in the same hierarchical level, which represents a

1 Specifies the available authentication mechanisms supported by a CSP on its offered Cloud services.
2 Specifies the level of assurance of the CSP supported mechanisms used to authenticate a user.
3 Specifies whether third party authentication is supported by the Cloud service.

direct horizontal dependency. Furthermore, $s_{2.1}$ depends on k_6 and k_7, which specifies direct vertical dependencies. As a result, $s_{2.2}$ depends on k_6 and k_7 indirectly through $s_2.1$ (i.e., Transitivity property). In many cases horizontal and vertical dependencies occur at the same time and both dependencies affect the whole composition hierarchy. We also consider different level of dependencies importance presented using a three level scale as shown in Table 2.1 and Figure 2.2.

All the dependencies explained so far are considered unidirectional dependencies. Other dependencies as bidirectional (interdependent relations between services) may occur as well. Bidirectional dependency occurs between services s_1 and s_2 if the provisioning of s_1 is conditional to the provisioning of s_2 and at the same time the provisioning of s_2 is conditional to the provisioning of s_1.

We assume that dependencies between services and SLOs in the secSLA are predefined and described by relevant standards working groups. In these groups, usually industry and academia, the secSLAs' contents are defined along with the type of dependencies and associated dependency importance levels. These sets of dependencies are categorized in the secSLA template. This template is later used in the creation of the dependency model and for the SLOs validation (cf., Section 2.3.4).

Numeric scale	Meaning
1	Weak dependency
2	Medium dependency
3	Strong dependency

Table 2.1.: Dependency importance level

2.3 QUANTITATIVE REASONING SYSTEM ARCHITECTURE

The system involves five progressive stages as depicted in Figure 2.3. In Stage (A), we express in a common way both the customer's security requirements and the CSP's committed SLOs using a standardised secSLA template (e.g., based on ISO/IEC 19086 [Int14]). The customer and CSPs' secSLAs from the preceding stage feeds into (B) to evaluate the rules and computes quantitative values of the customers' security requirements and CSPs' secSLA. A dependency model is created in Stage (C) to capture information about secSLAs' services and the dependencies that occur between them. This model is specified using a machine-readable format in order to allow automated validation for checking service conflicts and different SLOs' compatibility issues. After that in (D), the validated secSLAs are structured using the Dependency Structure Matrix (DSM). The structured secSLA shows the level of dependencies between services in a precise order that makes the secSLA clear. The structured secSLAs are utilized in Stage (E) to assess and rank the CSPs according to the customer requirements. We propose two assessment techniques which evaluate different security levels committed by various CSPs and rank them according to their secSLAs; namely, the Quantitative Hierarchical Process (QHP) [TTLS14] and the Fuzzy Quantitative Hierarchical Process (fuzzy-QHP) [TTLS17].

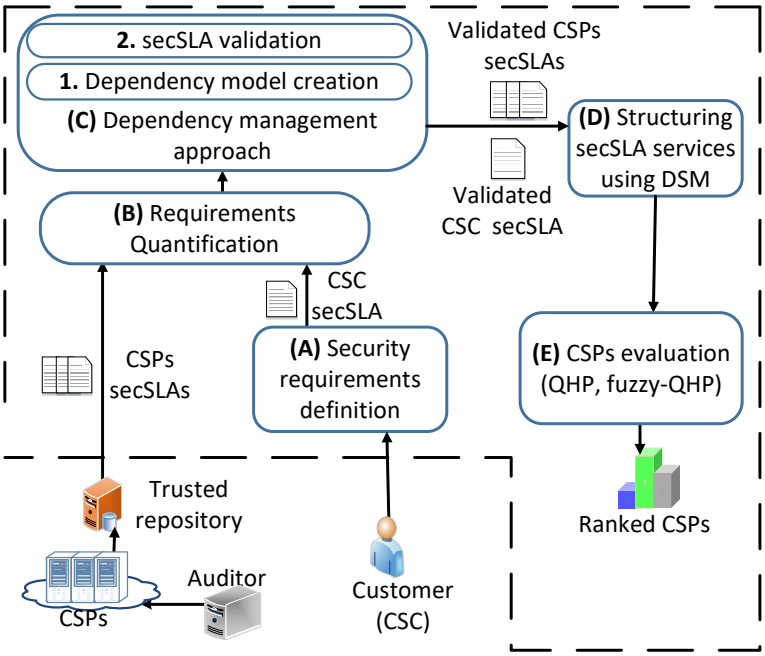

Figure 2.3.: Proposed system stages

2.3.1 *Trust Model*

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 the proposed system, the secSLAs provided by the participating CSPs are required to come from a trusted source. In a real-world setup, the trust relationships can be given by an *Auditor* performing a third-party attestation of the CSP secSLA (e.g., through a scheme such as the CSA Open Certification Framework (OCF) [Clo17c]). The audited secSLAs are then stored by the CSPs in a trusted repository of SLAs (e.g., the CSA STAR repository [Clo17d]), as shown in Figure 2.3. This trust assumption relies on the fact that the certifications and the repository are trusted, and that the published secSLAs were valid at the time of issuing the corresponding certification or publishing the information in the repository. Stronger assurance levels can be provided by mechanisms such as continuous monitoring (e.g., the CSA STAR continuous monitoring defined at the level 3 of the OCF). The described trust model is able to mitigate the risk of having malicious CSPs publishing false secSLA information with the goal of achieving higher scores in the proposed evaluation system. Other minor risks (e.g., tampered evaluation systems' software) can be mitigated through traditional security controls and secure software development techniques.

2.3.2 *Stage (A): Security Requirements Definition*

Customers can assess the CSPs and choose the CSP that best satisfies their security requirements only if the CSPs shared security information are relevant to customers' concerns (i.e., stemmed from customers' requirements) and are of an equal value [Meso6]. To achieve this, customer creates his/her set of security requirements using the same secSLA hierarchical structure used by the CSP to specify the offered security services (as depicted

in Figure 2.1)[4]. Accordingly, the Cloud secSLA hierarchy is used in two different ways. On the one hand, it is used to represent the CSP's secSLA by defining SLOs at the lowest level of the hierarchy. Each CSP's secSLA define the set of security services and their corresponding values that each CSP undertakes to fulfill. On the other hand, the secSLA hierarchy can also be used to define the customers' requirements and specify their preferences at different levels of granularity (which is at different levels of the hierarchy structure) according to their expertise.

Three different types of requirements specification, according to the customer expertise, are specified in this stage:

1. **Precise values**: Expert customers can specify their requirements at the SLO level by defining the required value for each SLO. Mainly, CSPs specify their offered levels at the SLO level.

2. **Linguistic terms**: Used by novice customers who could not specify precise values at the SLO level. In order to let these customers adopt Cloud services, it would be desirable to allow them to express their general requirements in a descriptive manner. Thus, customers specify qualitative requirements using terms such as (Not-Required (NR), Less-Required (LR), Required (R), Highly- Required (HR), and Do-not-know (Dk)) to reflect their requirements.

   Highly-Required denotes that all security SLOs are mandatory requirements for the customers. *Not-Required (NR*) indicates that the security SLOs are not required by the customers. *Required and Less-Required* specify the customers' different degrees of requirements importance where the customers can accept varied values specifying several degrees of importance that depend on the considered scale. *Do-not-know* specifies the customers' unknown requirements.

3. **Natural language statements**: Since humans are accustomed to using descriptive phrases to make rough estimations, they can easily submit meaningful requirements in the form of natural language statements. To achieve this, (i) such a language must be based upon information that customers find cognitively easy to reflect upon and express, and (ii) the information communicated in such language should be reasonably easy to interpret.

   Considering this list of requirements, it appears unavoidable that such a language would be based on linguistic terms or qualitative information. For example "I *highly require* a provider encrypting data in transit and at rest".

Blended submissions of different types of requirements for the same secSLA are also supported in our proposed approaches. Chapter 2.4 illustrates, using a real-world case study, how different types of requirements help customers to define their requirements. The output of this stage is a customer secSLA which is then used along with one or more CSP's secSLAs, as an input to the next stage.

2.3.3 *Stage (B): Requirements Quantification*

To assess and compare the security levels provided by different CSPs according to the different customers' expertise, a measurement model for different security SLOs is defined.

4 Cloud Security Alliance (CSA) has created a consensus assessments initiative questionnaire (CAIQ) [Clo17b], to define the security controls contained in an secSLA.

However, the process of modeling SLO values to a quantitative metric is not straightforward, as SLOs can have varied types/ranges of qualitative and quantitative values. Considering the different nature of SLOs (i.e., qualitative and/or quantitative), two assessment techniques are proposed in this chapter (QHP and fuzzy-QHP). Each assessment technique introduces a different approach to quantification.

For example, the QHP [TTLS14] defines security levels for each SLO separately, depending on the type of the SLO. QHP allows customers to specify precise values and/or linguistic terms to detail their requirements. Unlike QHP, fuzzy-QHP aims to solve the problem of "accurately" evaluating the security level of a CSP when qualitative requirements are defined by the customers. Fuzzy-QHP allows customers to specify their requirements using natural language statements as well as precise values and linguistic descriptors. These statements are defined using *if-then* rules which are used to capture the imprecision of customers' requirements according to their security expertise by using the triangular fuzzy numbers (TFNs) [Men95].

We detail the standalone operations of each technique from the secSLA perspective in Section 2.3.6.

2.3.4 *Stage (C): Dependency Management Approach*

A dependency model is created in this stage to capture information about secSLAs' 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 different SLOs' compatibility issues. The approach for managing service dependencies builds on a dependency model, which is used to capture information about security services and the dependencies that occur between them. In order to model service dependencies, it is important to first derive the expected requirements that the dependency model should support.

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

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

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

This dependency management approach is performed in two steps as shown in Figure 2.3.

Step 1: Dependency Model Creation

Handling all the dependencies in the secSLA is a very time consuming and complex task. Therefore, a dependency model is created for each secSLA to cover all identified dependencies within the secSLA. This model is used to capture information about services (each composed of a set of SLOs) and the dependencies that occur between them. We model a secSLA by a tuple $secSLA = (S, l, \rightarrow_S, K, \rightarrow_K, v)$ where:

- S is a set of services s with associated hierarchy levels $l(s) \in \{0, 1, \ldots, n-1\}$. As shown in Figure 2.2, a secSLA is composed of four hierarchical levels. A secSLA contains exactly one service s with $l(s) = 0$, which is the root service. Level n is the SLO level.

- $\rightarrow_S \subseteq S \times (S \cup K) \times \{1, 2, 3\}$ models service dependencies where $\{1, 2, 3\}$ shows the dependency importance level w (cf., Table 2.1).

- We write $s_1 \xrightarrow{w}_S s_2$ if s_1 (dependent service) depends on s_2 (antecedent service) with dependency importance level w, where $w \in \{1, 2, 3\}$. We write $s \rightarrow_S o$ to express that $s \xrightarrow{w}_S o$ for some $w \in \{1, 2, 3\}$ (where o is either a service or an SLO).

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

- $\rightarrow_K \subseteq K \times K \times \{1, 2, 3\}$ models SLO dependencies. We have $k_1 \xrightarrow{w}_K k_2$ if k_1 (dependent SLO) depends on k_2 (antecedent SLO) with importance level w.

- $v : K \mapsto V$ is an assignment of values in V to SLOs, where V is the set of all metric values of each SLO in K.

- Constraints on the SLO dependency relation are specified using a constraint set $C_v^{\rightarrow_K} \subseteq K \times K \times \{=, \neq, <, \leqslant, >, \geqslant\}$. A constraint $(k_1, k_2, \equiv) \in C_v^{\rightarrow_K}$ is satisfied if the values of k_1 and k_2 are related by the given comparison, i.e., $v(k_1) \equiv v(k_2)$. A dependency relation $k_1 \rightarrow_K k_2$ is called valid, written $valid_{C_v^{\rightarrow_K}}(k_1, k_2)$, if the relation satisfies all its constraints, i.e., $\forall(k_1', k_2', \equiv) \in C_v^{\rightarrow_K}.(k_1 = k_1' \text{ and } k_2 = k_2') \Rightarrow v(k_1) \equiv v(k_2)k$.

- We write the transitive closure of $\rightarrow_S$ as $\rightarrow_S^+$, i.e., $s_1 \rightarrow_S^+ s_2$ if $s_1 \rightarrow_S s_2$ or $\exists s_3 \in S.s_1 \rightarrow_S s_3$ and $s_3 \rightarrow_S^+ s_2$. A dependency $o_1 \rightarrow o_2$, where $\rightarrow \in \{\rightarrow_S, \rightarrow_K\}$, between two objects $o_1, o_2 \in S \cup K$ is called symmetric if also $o_2 \rightarrow o_1$. Otherwise, $o_1 \rightarrow o_2$ is called non-symmetric. In other words, o_1 and o_2 are symmetrically dependent if $o_1 \rightarrow o_2$ and $o_2 \rightarrow o_1$.
  Note that it is possible that $o_1 \rightarrow o_2$ is symmetric while the relation $\rightarrow$ is non-symmetric. We explain this using an example, let us consider $\rightarrow = \{(o_1, o_2, w), (o_2, o_1, w), (o_1, o_3, w)\}$ which is a non-symmetric relation and where we have that $o_1 \rightarrow o_2$ is symmetric.

- A secSLA has to satisfy the following constraints:
  1. Services do only depend on services of the same or the next lower hierarchy level: $\forall s_1, s_2 \in S.s_1 \rightarrow_S s_2 \Rightarrow l(s_1) = l(s_2)$ or $l(s_1) + 1 = l(s_2)$
  2. Only services of hierarchy level $n-1$ depend on SLOs: $\forall s \in S \forall k \in K.s \rightarrow_S k \Rightarrow l(s) = n - 1$
  3. Services do not depend on themselves: $\forall s \in S.\neg s \rightarrow_S s$
  4. All services depend directly or indirectly on an SLO: $\forall s \in S \exists k \in K.s \rightarrow_S^+ k$

Step 2: SecSLA Validation

A meta-model is developed based on the dependency definitions. This meta-model allows the description of SLOs along with information on the secSLA drafted for it. The meta-model is specified using a machine readable format (allowing an automaated validation)

such as an XML data structure using an XML Schema. In this Schema, dependency relations between services are modeled by including the involved service SLOs and their roles as dependent or antecedent as well as their values. Moreover, the constraint comparison is extracted and modeled in the Schema. A brief excerpt from the secSLA dependency model is shown in Listing 2.1, where an example of two SLOs named "User authentication and identity assurance level" (k_{Usauth}) and "CSP Authentication" (k_{CSauth}) and the dependent relation between them are shown, so that $k_{Usauth} \rightarrow_K k_{CSauth}$. Their security levels are modeled as $\nu(k_{Usauth})$ and $\nu(k_{CSauth})$, respectively.

The requirement is that the security level of Enc is higher than or equal to the security level of k_{Usauth}, i.e., $\nu(k_{CSauth}) \geqslant \nu(k_{Usauth})$. This requirement is modeled as $(k_{Usauth}, k_{CSauth}, \leqslant) \in C_v^{\rightarrow_K}$.
Note that all service levels (e.g., $level_2$, $level_3$, ...) are modeled as numerical values. These numerical values are the security SLOs' values in the XML schema shown in Listing 2.1.

Listing 2.1: Excerpt of dependency model of a secSLA

```
<secSLA slaid="sla1">
<dependencyModel Depmodelid="depmod_1">
        <sloDependency Depid="dep–1" type=‘‘unidirectional">
                <dependent depSLOid="AA1.1_sla1" value ="2"/>
                <antecedent antSLOid="AA1.2_sla1" value="3"/>
                <constraint> leq </constraint>
        </sloDependency>
</dependencyModel>
</secSLA>
```

Following the development of the dependency model, the secSLA SLOs are validated, as shown in Figure 2.4. The validation is done by first extracting the secSLA ID, dependency model ID, and dependency ID of two dependent SLOs (each dependency relation in the same dependency model has a unique ID) defined in the XML Schema. Furthermore, for each dependent relation, the antecedent and dependent SLO's values are extracted. This entails extracting the constraint comparative (i.e., $=, \neq, \leqslant, \geqslant$) and checking if the two dependent SLO values satisfy the constraint. If the constraint between dependent SLOs is not satisfied, the validation scheme shows a conflict between these two SLOs. The dependency ID and dependent SLO ID of the affected SLOs are saved in a list, while the evaluation is continued to determine further conflicts. At the end of this phase, a list of all conflicts found in the CSP's secSLA with the conflicts explanation is sent to the CSP in order to resolve these conflicts and resubmit his/her secSLA again to be validated. Similarly, a list of all conflicts found in the customer requirements (specified using the customer secSLA) are resolved by the customer and validated again. If no problems are detected, both the validated CSPs' secSLAs along with the customer's secSLA are used as an input to Stage (D).

Algorithm 1 details the necessary steps for extracting the values of validating a dependency relation between SLOs in pseudocode notation.

2.3.5 *Stage (D): Structuring SecSLA Services*

In a secSLA, as the number of offered services and SLOs along with the dependencies between them increases, the secSLA hierarchical structure (shown in Figure 2.2) quickly becomes cluttered and becomes a disorderly network of tangled arcs. This makes it hard

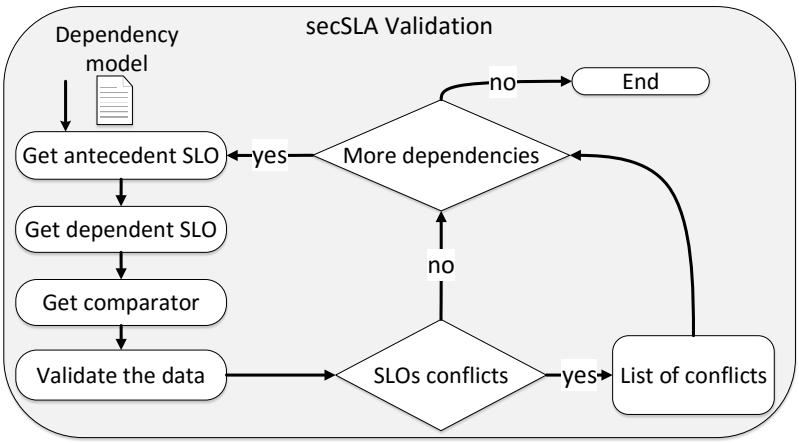

Figure 2.4.: Dependency based secSLA validation stages

for (i) the customers to specify their requirements and resolve the conflicts (which requires an expert customer and is time consuming) and (ii) the CSPs to check the dependency relations between their offered services to avoid any violation. Consequently, the objective of this stage is to embody the secSLA hierarchical structure by mapping the dependencies in a precise order where services and SLOs are ordered according to their level of dependency. This ordering makes the dependency relations explicit and more traceable regardless of the size which allows customers to (a) easily define their security requirements and (b) assess and rank the CSPs according to their security requirements. Furthermore, this provides CSPs with the guidance on the security improvements that should be performed in order to achieve the customer requested security level.

A variety of techniques exist for the analysis, management, and ordering of the secSLA services and SLOs other than the graphs used in building the secSLA hierarchical structure. One of these techniques is the program evaluation and review technique [WL77]. Although this technique incorporates more information than the directed graphs, it is still inadequate for representing the vast majority of design procedures where iteration[5] task relationships are involved. Another technique which has been widely used in documenting design procedures is the structured analysis and design technique [Ros77]. This technique attempts to overcome the size limitations by restricting the amount of information that can be placed on each document. Unfortunately, loops remain an unsolved problem [MM87].

A representation which overcomes the size and the iteration tasks limitations of those discussed above is the Design Structure Matrix (DSM) (also known as "Dependency Structure Matrix") [Ste81]. There are two main categories of DSMs: static and time-based [Bro01]. The DSM embodies the structure of the underlying design activity by mapping the relations between services in a precise order which makes the secSLA clear and easy to read, regardless of the size. To clarify, a secSLA of n services is represented as an $n \times n$ matrix with identical row and column labels. The matrix element a_{ij} is empty if the i^{th} column is independent on the j^{th} row, and not empty if they are dependent. This means, services and SLOs with empty rows have all required information and do not depend

5 A loop of information which occurs if there are bidirectional relations between services, which means each service is waiting for information from the other one.

Algorithm 1 : Validation

```
Input: String slaID, String depmodelID, String depID
Output: List affectedSLOs= NIL
List dependencyList = NIL
Value dependentValue = NIL
Value antecedentValue = NIL
List dependencyModels = getModels(depmodelID, slaID)
for model ∈ dependencyModels do
    dependencyList = getDependencies(depID, model)
    for depend ∈ dependencyList do
        dependentValue=getValue(depend.depSLOvalue, depID)
        antecedentValue=getValue(depend.antSLOvalue, depID)
        dependentConst=getConst(depend.constraint, depID)
        if validate(dependentValue, antecedentValue, dependentConst) ≠ true then
            affectedSLOs.add(depend.depSLO)
        end if
    end for
end for
```

on others. Furthermore, the empty columns provide no information required by other services and SLOs.

To demonstrate the idea of DSM, the mapping of the secSLA shown in Figure 2.2 into a DSM is presented in this section and is depicted in Table 2.2. As the dependencies of services on themselves are not considered (as specified in the dependency model constraints in Section 2.3.4), there are no marks along the diagonal. The strength of the dependencies is given using numerical values; these values provide more detailed information on the relationships between the different system services [Ste81]. We utilize the three level scale dependency importance rating defined in Table 2.1. Examining row 2, we note that s_1 strongly depends on s_2 and $s_{1.1}$, and weakly depends on $s_{1.2}$ and $s_{1.3}$. Examining row 10, we note that SLO k_2 weakly depends on k_3.

After mapping the secSLA into a DSM, we can start reordering the DSM rows and columns in order to transform the DSM into a lower triangular form (that is, the matrix has no entries above the diagonal), this is called DSM partitioning [GE91] and is done in two steps:

Step (1) Services which have a minimum number of dependencies (initially there will be none) are placed at the top of the DSM. These services are identified as services with minimum number of row values. If there is more than one such service, the one with maximum number of column values is selected.

Step (2) Services that deliver no information to others in the matrix are placed at the bottom of the DSM. This is easily identified by observing an empty column in the DSM. Once a service is rearranged, it is removed from the DSM and step 2 is repeated on the remaining elements.

Table 2.3 shows the result of partitioning the DSM depicted in Table 2.2. Bidirectional dependencies occur when the matrix cannot be reordered to have all matrix elements sub-diagonal. As shown in Table 2.3, k_2 and k_3 are bidirectionally dependent (indicated by

#		Root	s1	s2	s1.1	s1.2	s1.3	s2.1	s2.2	k1	k2	k3	k4	k5	k6	k7	k8	k9
1	**Root**	.	3	3														
2	**s1**		.	3	3	1	1											
3	**s2**			.		2		2	2									
4	**s1.1**				.					3	3							
5	**s1.2**					.						2	2					
6	**s1.3**					3	.	2					2	2		1		
7	**s2.1**							.							2	3	3	
8	**s2.2**					2		2	.								2	2
9	**k1**									.								
10	**k2**										.	1						
11	**k3**										1	.						
12	**k4**												.					
13	**k5**									2				.				
14	**k6**														.		1	
15	**k7**															.		
16	**k8**												2				.	
17	**k9**																	.

Table 2.2.: DSM mapping of the secSLA shown in Figure 2.2

shading); k_2 needs the information of k_3 and k_3 needs the information of k_2. If k_2 and k_3 are regarded as a single composite service, the cycle can be eliminated [SJSJ05].

Quantified and validated CSPs and customers secSLAs serve as an input for the next stage of the process (Stage (E)), where each CSP's secSLA is evaluated with respect to customer's secSLA.

2.3.6 *Stage (E): CSPs Evaluation*

The quantitative security level assessment of CSPs (for their match to the customer requirements) is the primary objective of the proposed framework developed in this stage. The challenge is not only how to quantify different SLOs in the secSLAs, but also how to aggregate them with a meaningful metric. Contemporary Cloud security assessment techniques mostly focus on their use in an environment where performance is not important (e.g., in decision-making dashboards where the customer requirements are elicited for the entire secSLA and only subsequently evaluated). Nevertheless, it is important to develop high efficiency assessment algorithms which, by decreasing the time complexity of each assessment cycle, (i) enable processing of many requests in parallel, and (ii) facilitate customer decisions by allowing them to select and adjust their requirements on the fly according to the current results based on what has already been chosen.

To this end, we have developed two assessment techniques to evaluate security levels guaranteed by CSPs and rank them by quantifying and comparing their secSLAs. In this section, we detail each of the proposed assessment techniques. We first detail the

#		k1	k4	k7	k9	k3	k2	k8	k6	k5	s1.2	s1.1	s2.1	s2.2	s1.3	s2	s1	Root
1	k1	.																
2	k4		.															
3	k7			.														
4	k9				.													
5	k3					.	1											
6	k2					1	.											
7	k8		2					.										
8	k6							1	.									
9	k5	2								.								
10	s1.2		2			2					.							
11	s1.1	3					3					.						
12	s2.1			3				3	2				.					
13	s2.2				2			2			2		2	.				
14	s1.3		2	1						2	3		2		.			
15	s2										2		2	2		.		
16	s1										1	3			1	3	.	
17	Root															3	3	.

Table 2.3.: Final DSM mapping of the secSLA shown in Figure 2.2 after partitioning and scheduling

standalone operations of each technique from the secSLA perspective, and subsequently discuss guidance for their usage discretely and collectively. Also the empirical validation of these techniques will be presented through four use cases (using real-world data) in Section 2.4. As an overview of the two techniques, the secSLA assessment and the ranking of CSPs is performed in four progressive phases as depicted in Figure 2.5.

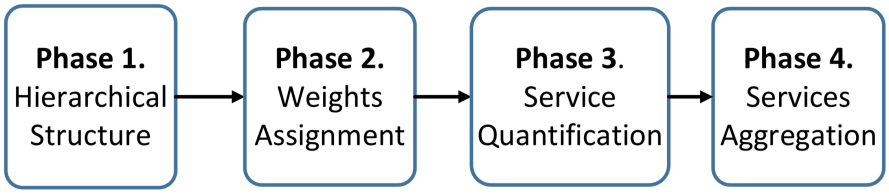

Figure 2.5.: CSPs evaluation process

2.3.6.1 *Quantitative Hierarchy Process (QHP)*

Given the fact that a secSLA might have a high number of individual security SLOs and that customers might specify their requirements with different levels of granularity, QHP is based on Analytic Hierarchy Process (AHP) [Saa90] for solving Multiple Criteria Decision Making (MCDM) [Zel82] problems. The advantages of AHP over contemporary

methods are its ability to handle composite qualitative and quantitative attributes, along with its flexibility and ability to identify inconsistencies across requirements [Saa90].

QHP is developed in the following progressive phases as depicted in Figure 2.5.

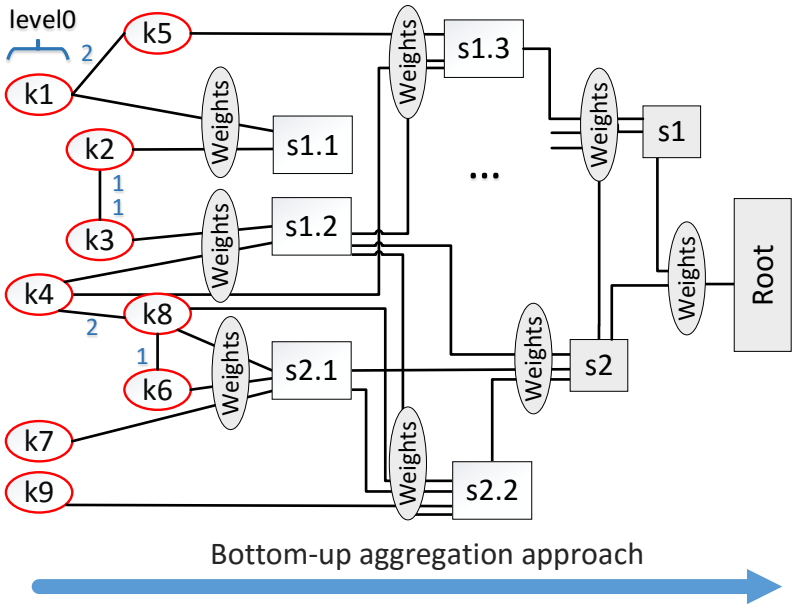

Figure 2.6.: Dependent secSLA Hierarchy

Phase 1. Hierarchical Structure

The DSM mapping of each secSLA (either the customer's secSLA or the CSPs' secSLAs) is modeled as a hierarchical structure in this phase. In this structure, the top level of the hierarchical structure defines the main goal and aims to find the overall rank (i.e., the root level). The lowest level is represented by the least dependent SLOs. Figure 2.6 outlines the hierarchical representation of the DSM shown in Table 2.3, where lowest level is represented by the least dependent SLOs which are the SLOs with empty rows in Table 2.3 . Using hierarchical structure, basic customers can simply specify their requirements at the lowest level only (i.e., $level_0$ in Figure 2.6).

Phase 2. Weights Assignment

The dependency importance level of each dependency relation (strong, medium or weak dependency with numerical scaling 3, 2 and 1, respectively; cf., Table 2.1) is specified as a weight (w) during the assessment process. The weight of each relation is used to show how different dependency levels affect the overall security level.

Phase 3. Service Quantification

In order to evaluate the Customer requirements with respect to a CSP's secSLA, the so-called measurement model for different security SLO metrics needs to be defined. Over this phase, different comparison metrics for different types of requirements are defined, so they can be applied for the quantitative security assessment.

Denoting CSP_i as the i-th CSP, $i = 1, 2, \ldots, m$, and k as the k-th SLO, $k = 1, 2, \ldots, n$. Notations presented in Table 2.4 are used in this section.

Term	Definition
V	The set of all values of each SLO (k) in the set of SLOs (K)
$v_{\mathsf{CSP},k}$	CSP provided value for SLO k.
$v_{\mathsf{CSC},k}$	Customer CSC required value for SLO k.
$W_{i,j,k}$	Relative rank of CSP_i over CSP_j with respect to SLO k.
$W_{i,\mathsf{CSC},k}$	Relative rank of CSP_i over customer CSC with respect to SLO k.

Table 2.4.: QHP terms

Security SLOs considered in the secSLA can be either *boolean* (expressing whether a CSP offers a security feature or not, e.g., encryption of data at rest) or *numerical* (expressing different possible values for a security property, e.g., cryptographic key length). The QHP method handles both cases by modeling them with security levels. Assuming a numerical SLO k can have n different values $v_1, v_2, \ldots, v_n$, where value v_n assures the highest level of security with respect to the SLO k, the values are modeled as $v_i \to i$, $i = 1, 2, \ldots, n$. Let us assume that the highest security level assigned to all numerical SLOs considered in the assessment process equals N. In this case, boolean metrics are modeled as $yes \to N$ and $no \to 0$ assuming yes provides higher security assurance and $no \to N$ and $yes \to 0$ if no assures better security service.

Note that QHP assumes that if a CSP offers a specific value for an SLO, it is also able to provide all SLO values with lower level of granted security. The QHP does not differentiate between a CSP that is able to grant the exact provision required by the customer (CSC) and a CSP that is able to offer even higher security levels. Thus, before any calculations are done, all SLO values for all CSPs are normalized to the customer requirements (i.e., each $v_{i,j}$ is updated to $\min(v_{i,k}, v_{\mathsf{CSC},k})$). This setting eliminates the so-called *masquerading effect* which occurs when the overall aggregated security level value mostly depend on those security controls with a high-number of SLOs, thus affecting negatively groups with fewer although possibly more critical provisions.

The QHP ranks CSPs by performing pairwise comparisons among all of them. The *relative rank* of CSP_i over CSP_j with respect to the SLO k is defined as

$$W_{i,j,k} = \begin{cases} 1, & \text{if } v_{i,k} = 1, \\ 0, & \text{if } v_{i,k} = 0, \end{cases} \tag{2.1}$$

in boolean case, and as

$$W_{i,j,k} = \begin{cases} 1, & \text{if } v_{i,k} \equiv v_{j,k}, \\ v_{i,k}/v_{j,k}, & \text{if } v_{i,k} \neq v_{j,k}, \end{cases} \tag{2.2}$$

in numerical case. The same formulas hold when evaluating a CSP with respect to the CSC desired value for an SLO. In the boolean case, we take

$$W_{i,CSC,k} = \begin{cases} 1, & \text{if } v_{i,k} = 1, \\ 0, & \text{if } v_{i,k} = 0. \end{cases} \tag{2.3}$$

In the numerical case, the expression becomes

$$W_{i,CSC,k} = \begin{cases} 1, & \text{if } v_{i,k} \equiv v_{CSC,k}, \\ v_{i,k}/v_{CSC,k}, & \text{if } v_{i,k} \neq v_{CSC,k}. \end{cases} \tag{2.4}$$

In order to rank CSPs for a specific SLO k with respect to CSC desired value for the SLO, the QHP methodology defines the *Comparison Matrix* (CM) as

$$CM_k = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ \vdots \\ CSP_n \\ CSC \end{array} \begin{array}{c} \begin{array}{cccccc} CSP_1 & CSP_2 & \dots & CSP_n & CSC \end{array} \\ \begin{pmatrix} W_{1,1,k} & W_{1,2,k} & \dots & W_{1,n_{CSP},k} & W_{1,CSC,k} \\ W_{2,1,k} & W_{2,2,k} & \dots & W_{2,n_{CSP},k} & W_{2,CSC,k} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W_{n_{CSP},1,k} & W_{n_{CSP},2,k} & \dots & W_{n,n_{CSP},k} & W_{n_{CSP},CSC,k} \\ W_{CSC,1,k} & W_{CSC,2,k} & \dots & W_{CSC,n_{CSP},k} & W_{CSC,CSC,k} \end{pmatrix} \end{array} \tag{2.5}$$

The relative ranking of all CSPs for the SLO k is defined as the normalized eigenvector (called a *Priority Vector* (PV)) of the corresponding comparison matrix CM_k. The PV indicates the numerical ranking of all the CSPs by specifying an order of preference among them, as indicated by the ratios of the numerical values.

Phase 4. Services Aggregation

In the final phase, we follow up with a bottom-up aggregation to give an overall assessment of the security levels and a final ranking of the CSPs (cf., Figure 2.6). To achieve that, the priority vector of each SLO (Phase 3) is aggregated with their relative normalized weights (dependency importance level) specified in Phase 2. This aggregation process is repeated for all the SLOs in the hierarchy with their relative weights.

$$PV_{aggregated} = \begin{bmatrix} PV_{k_1} & \dots & PV_{k_n} \end{bmatrix} \cdot \begin{bmatrix} NW \end{bmatrix}^T \tag{2.6}$$

Where NW is the set of normalized weights of different SLOs such that $NW = nw_{k_1}, nw_{k_2}, \dots, nw_{k_n}$. Note that the weights are normalized to satisfy the AHP requirements. PV_{k_1} is the PV calculated for SLO k_1. $PV_{aggregated}$ is the aggregated PV which shows the ranking of all the CSPs based on the customer-defined requirements and weights. We demonstrate and validate the framework presented in this section using a real-world case study in Section 2.4.

2.3.6.2 *Fuzzy Quantitative Hierarchical Process (Fuzzy-QHP)*

The assessment approach is utilized so that, the CSPs are ranked (as per their secSLA's) for the best match to the customer requirements. As specified earlier, the challenge is not only how to quantify different secSLA services, but also how to aggregate them in a

meaningful metric. Multiple Criteria Decision Making (MCDM) [ZC73] methods such as the Analytic Hierarchy Process (AHP) [Saa90] are used to solve these issues, as specified in QHP. However, according to [KH11], the AHP method is: (1) mainly used in nearly fixed decision applications and (2) it is not able to reflect the decision makers' uncertain preferences through fixed values.

Fuzzy-AHP is used to relieve the uncertainty and inability of the AHP in handling uncertain preferences. It allows a more accurate description of the decision-making process, where fuzzy set theories are used to express the uncertain requirements and preferences as fuzzy numbers. Basically, a fuzzy number defines a fuzzy interval in the real number, in the sense that it does not refer to one single value but rather to a connected set of possible values, where each possible value has its own membership function between 0 and 1. Hence, fuzzy numbers are used to translate the vagueness and imprecision of customers' requirements according to their security expertise. The most commonly used shapes for fuzzy numbers are triangular, trapezoidal, piecewise linear and Gaussian. The triangular fuzzy numbers (TFNs) [Men95] (depicted in Figure 2.7) are used in our study to capture the vagueness of the parameters by representing the fuzzy interval by two end points and a peak point (l, m, u), where the parameters l and u represents the two end points and denote the lowest possible value, and the upper possible value respectively. While m denotes the most promising value that describes the fuzzy event (i.e., when $l = m = u$, the fuzzy number becomes a real number). We refer the reader to Appendix A.1 for further details of TFN $\tilde{M}$.

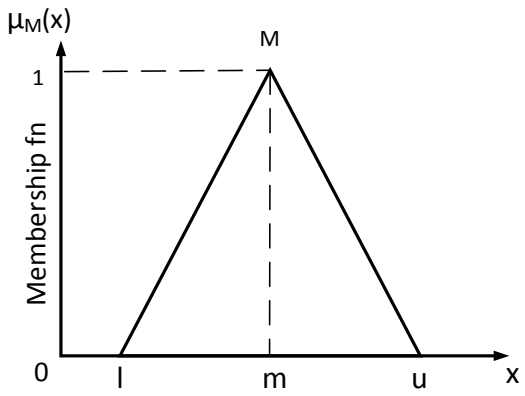

Figure 2.7.: Triangular fuzzy number.

As an overview of our evaluation framework, the secSLA assessment and the ranking of CSPs are performed in the following progressive phases:

Phase 1. Hierarchical Structure

Similarly, as in QHP, the DSM mapping of each secSLA is modeled as a hierarchical structure from the least to the most dependent services.

Phase 2. Weights Assignment

Similarly, as in QHP, the dependency importance level of each dependency relation (i.e., strong, medium or weak dependency with numerical scaling 3, 2 and 1, respectively).

Phase 3. Services Quantification

In this phase, three cases for the TFN representation of the SLO values are considered. These cases are considered according to the type of customer requirements as specified Section 2.3.2.

1. **Linguistic terms:** are represented as TFNs and specified using their membership functions as depicted in Figure 2.8.

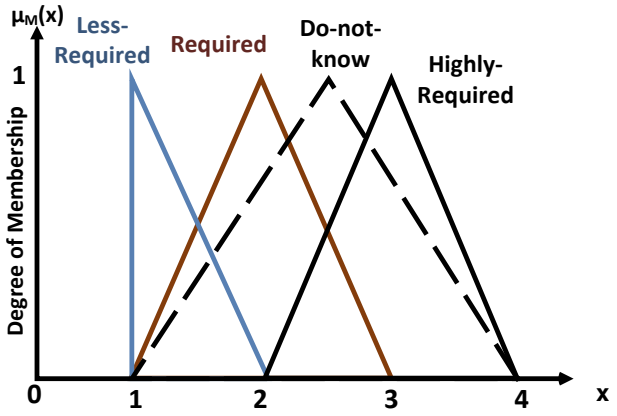

Figure 2.8.: Linguistic terms for criterion importance.

The proposed methodology allows the customers to: i) assign linguistic terms at varied levels of the hierarchical specification, and ii) individually adjust the linguistic terms according to their requirements. To further ease the task, especially for novice customers, the system can set default values for each linguistic requirement as shown in Figure 2.8 and Table 2.5. In our model, we set the range of membership functions between 0 and 4, but as specified earlier, this range is up to the user definition.

We represent *Not-Required (NR)* as $(0, 0, 0)$ as it is not required by the customer and *Do-not-know* which specifies customer uncertain requirements as a TFN that can have all possible ranges from 1 to 4, we define it as $(1, 2.5, 4)$, which means the most promising value is 2.5.

Linguistic scale for importance	**TFN (l, m, u)**
Less-Required (LR)	$(1, 1, 2)$
Required (R)	$(1, 2, 3)$
Highly-Required (HR)	$(2, 3, 4)$
Not-Required (NR)	$(0, 0, 0)$
Do-not-know (Dk)	$(1, 2.5, 4)$

Table 2.5.: Linguistic terms and values of ratings.

2. **Natural language statements:** are mapped into values representing the customer requirements. This can be done only if we have a consistent set of statements for a

given language. In that case, every statement is mapped to the appropriate parameter that yields to a value function that represents the customer requirement. We define this set of statements using *if-then* rules, where the customer should specify the required *service name* and his/her *preferences* in his/her statement. Using a fuzzy search algorithm[6] based on Levenshtein distance, we get the required service from the list of secSLA services and the needed preference. For example, "I highly require a provider encrypting data in transit and at rest" statement is converted to an *if-then* rule at the end of this stage, such that: **If** *data encryption in transit* (s_1) is *highly required* AND *data encryption at rest* (s_2) is *highly required* **then** $\tilde{v}_{s_1}$ is *HR* AND $\tilde{v}_{s_2}$ is *HR*. Thus, at the end of this stage the customer requirements are assigned as linguistic terms and represented using TFNs. Multiple rules specifying different security controls can be combined using AND or OR operators. In addition, if a customer does not specify or does not care about other services, by default these services are set to *Less-Required*.

3. **Precise values** assigned as fixed values to each SLO k_i.

   We demonstrate that using a real-world case study in Section 2.4.

To formalize the usage of TFNs in order to model the customers vague requirements, we extend Definition 1 by the following definition:

Definition 2 *Each SLO* k_i *in a secSLA consists of* m *different values* $\tilde{v}_i$. *Each value implies a different security level offered by the CSP and required by the customer. Each value* $\tilde{v}_i$ *is specified using TFN* (l_i, m_i, u_i)*; such that* $k_i = \{\tilde{v}_{i,1}, \tilde{v}_{i,2}, \ldots, \tilde{v}_{i,j}\}$. *The total order of security levels is defined using an order relation "*$\prec_i$*"; such that* $k_i = \tilde{v}_{i,1} \prec \tilde{v}_{i,2} \prec \ldots \prec \tilde{v}_{i,j}$.

In order to assess each CSP's secSLA, a quantification model for different SLOs should be defined. We use the fuzzy-AHP relative ranking model based on a pairwise relation of the services (a) provided by different CSPs, and (b) required by customers' such that:

$$CSP_{1,k_i}/CSP_{2,k_i} = \frac{\tilde{v}_{1,k_i}}{\tilde{v}_{2,k_i}} \tag{2.7}$$

where $CSP_{1,k_i}/CSP_{2,k_i}$ indicates the relative rank of CSP_1 over CSP_2, regarding k_i. such that:

$$\begin{aligned} CSP_{1,k_i}/CSP_{2,k_i} &= (1,1,1), && \text{if } \tilde{v}_{1,k_i} \equiv \tilde{v}_{2,k_i} \\ &= \frac{(l_1, m_1, u_1)}{(l_2, m_2, u_2)}, && \text{if } \tilde{v}_{1,k_i} \not\equiv \tilde{v}_{2,k_i} \end{aligned}$$

The TFN division is defined as [PSK08]:

$$\frac{(l_1, m_1, u_1)}{(l_2, m_2, u_2)} = (l_{12}, m_{12}, u_{12}) = \left(\frac{l_1}{u_2}, \frac{m_1}{m_2}, \frac{u_1}{l_2}\right) \tag{2.8}$$

Similarly, $CSP_{i,k}/CSC_k$ indicates the relative rank of CSP_i over Cloud Service Customer CSC, which specifies whether or not CSP_i satisfies the customer requirements, with respect to k. This pairwise comparison results in a one-to-one comparison matrix $\tilde{A}$ for each

6 Fuzzy search algorithm as well as basic notations used covering fuzzy and crisp sets, membership functions, operations of TFN are specified in Appendix A.1

SLO. The comparison matrix is of size $(n+1) \times (n+1)$, considering n CSPs and one CSC for each SLO, such that:

$$\tilde{A}_{k_i} = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ n \\ n+1 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & \dots & n & n+1 \end{array} \\ \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \dots & \tilde{a}_{1n} & \tilde{a}_{1u} \\ \tilde{a}_{21} & \tilde{a}_{22} & \dots & \tilde{a}_{2n} & \tilde{a}_{2u} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \dots & \tilde{a}_{nn} & \tilde{a}_{nu} \\ \tilde{a}_{u1} & \tilde{a}_{u2} & \dots & \tilde{a}_{un} & \tilde{a}_{uu} \end{pmatrix} \end{array} \tag{2.9}$$

where $\tilde{a}_{ij} = CSP_i/CSP_j$, $\tilde{a}_{iu} = CSP_i/CSC$. For example, $\tilde{a}_{12} = CSP_1/CSP_2$ regarding SLO k_i, which indicates the relative rank of CSP_1 over CSP_2. Note that, $\tilde{a}_{ij} = \frac{1}{\tilde{a}_{ji}} = (\frac{1}{u_{ji}}, \frac{1}{m_{ji}}, \frac{1}{l_{ji}})$ (cf., Equation 2.8).

Next, the respective scores for all the CSPs and the customer, for each SLO, are obtained by calculating the priority vector (PV) of the corresponding fuzzy comparison matrix $\tilde{A}_{k_i}$. There are several procedures to attain PV in fuzzy-AHP. The widely used Chang's extent analysis method [Cha96] is the one utilized in our technique (fuzzy-QHP).

Chang's extent analysis method on fuzzy-AHP overview

We explain Chang's method using an example of two TFNs (l_1, m_1, u_1) and (l_2, m_2, u_2) which represent a provider security-level and a user requirement for a particular SLO. By calculating the comparison matrix using Equation 2.9, the comparison matrix is equal to:

$$\tilde{A} = \begin{array}{c} \\ i=1 \\ i=2 \end{array} \begin{array}{c} \begin{array}{cc} j=1 & j=2 \end{array} \\ \begin{pmatrix} (1,1,1) & (l_{12}, m_{12}, u_{12}) \\ (l_{21}, m_{21}, u_{21}) & (1,1,1) \end{pmatrix} \end{array}$$

After $\tilde{A}$ calculation, the steps of Chang's extent analysis to obtain the PV are detailed as follows:

- **Step 1.** The value of the fuzzy synthetic extent with respect to ith object is calculated such that:

$$E_i = (\sum_{j=1}^{m} l_j, \sum_{j=1}^{m} m_j, \sum_{j=1}^{m} u_j) \otimes (\frac{1}{\sum_{i=1}^{n} u_j}, \frac{1}{\sum_{i=1}^{n} m_j}, \frac{1}{\sum_{i=1}^{n} l_j}) \tag{2.10}$$

  Whereas $\otimes$ denotes fuzzy multiplication, $i = 1 \dots n$, and $j = 1 \dots m$ so that:

$$E_i = (\frac{\sum_{j=1}^{m} l_j}{\sum_{i=1}^{n} u_j}, \frac{\sum_{j=1}^{m} m_j}{\sum_{i=1}^{n} m_j}, \frac{\sum_{j=1}^{m} u_j}{\sum_{i=1}^{n} l_j})$$

  We explain this step using the two considered TFNs' comparison matrix $\tilde{A}$ ($m = n = 2$) so that:

$$E_1 = (1 + l_{12}, 1 + m_{12}, 1 + u_{12}) \otimes (\frac{1}{1 + u_{12} + 1 + u_{21}}, \frac{1}{1 + m_{12} + 1 + m_{21}}, \frac{1}{1 + l_{12} + 1 + l_{21}})$$

$$E_2 = (l_{21}+1, m_{21}+1, u_{21}+1)\otimes$$
$$(\frac{1}{1+u_{12}+1+u_{21}}, \frac{1}{1+m_{12}+1+m_{21}}, \frac{1}{1+l_{12}+1+l_{21}})$$

By the end of this step, E_1 and E_2 will be represented as TFN with values (l_1, m_1, u_1) and (l_2, m_2, u_2).

- **Step 2.** The degree of possibility of $E_2 = (l_2, m_2, u_2) \geqslant E_1 = (l_1, m_1, u_1)$ is defined as $D(E_2 \geqslant E_1) = sup[min(\mu_{E_1}(x), \mu_{E_2}(x))]$ and is represented as follows:

$$D(E_2 \geqslant E_1) = \begin{cases} 1, & \text{if } m_2 \geqslant m_1 \\ 0, & \text{if } l_1 \geqslant u_2 \\ \dfrac{l_1 - u_2}{(m_2 - u_2)(m_1 - l_1)}, & \text{otherwise} \end{cases} \tag{2.11}$$

For the comparison we need the values of both of $D(E_1 \geqslant E_2)$ and $D(E_2 \geqslant E_1)$.

- **Step 3.** The degree possibility for a fuzzy number to be greater than o fuzzy numbers E_i where $i = 1, 2, \ldots, o$ can be defined by
$D(E \geqslant E_1, E_2, \ldots, E_o) = D[(E \geqslant E_1), (E \geqslant E_2), \ldots, (E \geqslant E_o)]$

$$D(E \geqslant E_1, E_2, \ldots, E_o) = min(D(E \geqslant E_i)), i = 1, 2, \ldots, o \tag{2.12}$$

Assuming that $d'(A_i) = min(D(E_i \geqslant E_o))$, for $o = 1, 2, \ldots, n$; $o \neq i$. Then the priority vector is given by $PV_0 = (d'(A_1), d'(A_2), \ldots, d'(A_n))^T$ where $A_i (i = 1, 2, \ldots, n)$ are n elements.

- **Step 4.** Via normalization, the normalized priority vectors are $PV = (d(A_1), d(A_2), \ldots, d(A_n))^T$ where PV is a non-fuzzy number that gives priority weights of an attribute with respect to other attributes.

At the end of Step 4, we attain the priority vector of the fuzzy comparison matrix for a particular SLO. This method is done for all the SLOs' matrices.

$$PV_{k_i} = \begin{array}{c} \begin{array}{ccccc} CSP_{1,k_i} & CSP_{2,k_i} & \ldots & CSP_{n,k_i} & CSC_{k_i} \end{array} \\ \begin{pmatrix} N_{1,k_i} & N_{2,k_i} & \ldots & N_{n,k_i} & N_{CSC,k_i} \end{pmatrix} \end{array} \tag{2.13}$$

Such that N_{1,k_i} is a numerical value representing the relative rank of CSP_1 to other CSPs and the CSC regarding SLO k_i. Similarly, N_{CSC,k_i} is the relative rank of the CSC required security level with respect to the security levels offered by the CSPs.

Phase 4. Services Aggregation

In the final step of the QHP process, the evaluation of the overall security level offered by each CSP (and thus the final ranking of CSPs with respect to customer requirements) is obtained by the bottom-up aggregation. This means that a PV is calculated for each element in the secSLA considering the importance weight assigned to the element by the customer. For n elements with importance weights nw_i, $i = 1, 2, \ldots, n$, to be aggregated at some hierarchy level (either at SLO, control group or control category level) we have

$$PV_{aggregated} = \begin{bmatrix} PV_{k_1} & \ldots & PV_{k_n} \end{bmatrix} \cdot \begin{bmatrix} nw_{k_1}, nw_{k_2}, \ldots, nw_{k_n} \end{bmatrix}^T \tag{2.14}$$

$PV_{aggregated}$ represent the scores for the entire secSLAs. The last element of the $PV_{aggregated}$ is the score for the customer secSLA and serves as the benchmark. Note that the weights are normalized to satisfy the fuzzy-AHP requirements.

2.4 CASE STUDY: SECURITY EVALUATION OF CSP'S SECSLAS

This section shows an empirical validation of the proposed framework through four scenarios that use real-world secSLA information derived from the CSA-STAR repository [Clo17d].

Cloud secSLA based on CSA STAR [Clo17d]									CSPs			Customer (CSC)				
Services						SLO			CSP_1	CSP_2	CSP_3	Case I		Case II	Case III	Case IV
category	lvl	category	lvl	category	lvl	name	dep.	lvl				req	rev			
Root	3	Audit & Compliance AC	3	Planning AC1	2	AC1.1	Dep_1		yes	yes	yes	yes		yes	HR	HR
					3	AC1.2	Dep_2	2	$level_2$	$level_2$	$level_3$	$level_3$				
			3	Independent Audits AC2	3	AC2.1	Dep_1	2	yes	yes	yes	yes			HR	
							Dep_3	3								
					3	AC2.2	Dep_4	3	no	yes	yes	no	yes			
					2	AC2.3	Dep_4		no	yes	yes	yes				
							Dep_5	2								
					1	AC2.4	Dep_6	1	yes	no	yes	yes				
					1	AC3.1			yes	yes	yes	yes				
					1	AC3.2			no	yes	yes	yes				
			3	Regulatory Mapping AC3	2	AC3.1	Dep_3		yes	yes	yes	yes		yes	NR	
					2	AC3.2	Dep_5		no	yes	yes	yes		yes		
					3	AC3.3			$level_2$	$level_1$	$level_3$	$level_3$		$level_3$		
	3	Business Continuity BC	2	Testing BC2	1	BC2.1	Dep_7	1	yes	yes	yes	no	yes		Dk	NR
					1	BC2.2	Dep_6		no	yes	no	no		no		
			2	Policy BC11	3	BC11.1	Dep_7		yes	yes	yes	yes		yes		
					3	BC11.2			$level_3$	$level_2$	$level_3$	$level_3$		$level_3$		
			2	Regulatory Mapping AC3	2	AC3.1			yes	yes	yes	yes				
					2	AC3.2			no	yes	yes	yes				
					3	AC3.3			$level_2$	$level_1$	$level_3$	$level_3$				
	3	Interface Security IS	3	Application Security IS1	2	IS1.1	Dep_8	1	weekly	weekly	daily	weekly	daily	daily	daily	
					1	IS1.2	Dep_8	1	$level_2$	$level_2$	$level_3$	$level_3$		$level_3$	$level_3$	
							Dep_2									
					1	AC3.1			yes	yes	yes	yes				

Table 2.6.: Case Study: Excerpt of secSLA from CSPs and customer requirements.

2.4.1 *The Customer Perspective: Security Comparison of CSPs*

This initial validation scenario, demonstrates how a Cloud customer can apply the presented framework to compare side-by-side three different CSPs based on their advertised secSLAs (compliant with the hierarchy of Figure 2.2) and with respect to a particular set of security requirements (also expressed as a secSLA). Table 2.6 presents a sample dataset used for this scenario which is based on the information available in the CSA STAR repository, where the values associated to 15 SLOs 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 security levels (cf., Definition 1) such as $monthly, weekly$, and $daily$, denoted as security levels $level_1, level_2, level_3$, and then modeled as TFN values $\tilde{1}, \tilde{2}, \tilde{3}$ (cf., Definition 2). Furthermore, no and yes are denoted as $\tilde{0}$ and $\tilde{3}$, respectively.

All CSPs' security SLOs are normalized to the customer requirements to eliminate masquerading. Furthermore, Table 2.6 shows four sets of Cloud customer requirements used as a baseline for comparing the selected CSPs. The first two cases specify a customer specifying precise SLO values, thus QHP and fuzzy-QHP are used to rank CSPs. The other two cases model novice customer with uncertain requirements, hence only fuzzy-QHP is used to rank the CSPs according to the customer requirements.

1. In Table 2.6 column marked as "Case I", the customer requirements are expressed at a per-SLO granular level. This represents a security-expert customer where the customer specifies his/her requirements at Case I "*req*" column. After that the customer secSLA is validated to check if each constraint between two dependent SLOs is satisfied and at the end of the validation showing the conflicts found. The customer then resolves the SLO conflicts by specifying new values to the SLOs causing conflicts. These new values are shown at Case I "*rev*" column in Table 2.6.

2. The column marked as "Case II" shows a set of requirements on SLOs that do not depend on any other SLO (identified using the DSM). This set of SLOs is used to model the customer requirements for the remaining SLOs. This might be the case of a non-expert customer who cannot specify all the secSLA SLOs and resolve the SLO conflicts, if any are found.

3. Column "Case III" shows a customer specifying (i) linguistic terms at different levels of the hierarchical structure according to his/her expertise with each service and (ii) detailed specification for other SLOs at the lowest level. Linguistic terms are specified as TFN as depicted in Figure 2.8. This case represents a semi-expert customer.

4. The column marked as "Case IV", customer requirements are specified using natural language statements. This might be the case of a customer that is not a security expert (i.e., novice customer).

Finally, we consider dependencies between services and SLOs which are going to be validated using the validation model presented in Section 2.3.4 such that:

SLO dependencies:

- AC2.1 is *medium* dependent on AC1.1 (this dependency relation is named as Dep_1 in Table 2.6 and the level of dependency is shown in the SLO "*lvl*" column). This is modeled as $AC2.1 \xrightarrow{2}_{K} AC1.1$ with constraint $(AC2.1, AC1.1, =) \in C_v^{\rightarrow_K}$.

- AC1.2 is *medium* dependent on IS1.2 (i.e., named as Dep_2) so that AC1.2 $\xrightarrow{2}_K$ IS1.2 with constraint $(AC1.2, IS1.2, =) \in C_v^{\rightarrow_K}$.
- In the same way, Dep_3, Dep_4 and Dep_5 are specified as AC2.1 $\xrightarrow{3}_K$ AC3.1 with $(AC2.1, AC3.1, =) \in C_v^{\rightarrow_K}$, AC2.2 $\xrightarrow{3}_K$ AC2.3 with $(AC2.2, AC2.3, =) \in C_v^{\rightarrow_K}$ and AC2.3 $\xrightarrow{2}_K$ AC3.2 with $(AC2.3, AC3.2, =) \in C_v^{\rightarrow_K}$ respectively.
- Dep_6 is modeled as AC2.4 $\xrightarrow{2}_K$ BC2.2 with constraint $(AC2.4, BC2.2, \neq) \in C_v^{\rightarrow_K}$.
- Finally, IS1.1 and IS1.2 are symmetrically dependent (i.e., Dep_8) so that IS1.1 $\xrightarrow{1}_K$ IS1.2 $\wedge$ IS1.2 $\xrightarrow{1}_K$ IS1.1 with constraint $(IS1.1, IS1.2, =) \in C_v^{\rightarrow_K}$.

Service dependencies:

- AC equally depends on AC1, AC2 and AC3 (i.e., equally depends refers to the level of dependency shown in the second column named "*lvl*" in Table 2.6). So that AC $\xrightarrow{3}_S$ AC1 $\wedge$ AC $\xrightarrow{3}_S$ AC2 $\wedge$ AC $\xrightarrow{3}_S$ AC3.
- BC equally depends on BC2, BC11 and AC3. Furthermore, IS *strongly* depends on IS1. Each is further depending on the SLOs with different dependency importance levels as shown in Table 2.6.
- AC2 depends on two of AC3 SLOs which are AC3.1 and AC3.2 (shaded in Table 2.6).
- 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 2.6).

2.4.1.1 *Cloud customer Case I requirements: Expert customer*

In this case, both the customer requirements and the CSPs secSLAs are validated to check conflicts between SLOs based on the defined dependencies as specified in Section 2.3.4. Errors in the customer requirements are automatically detected based on the modeled dependencies, such that:

- Dep_1 **Validation**: AC2.1 security level ($level_1$) is *equal* to the AC1.1 security level ($level_1$), $v(AC2.1) = v(AC1.1)$. **Result:** Valid.
  Dep_2 **Validation**: AC1.2 security level ($level_3$) is *equal* to IS1.2 security level. **Result:** Valid. Similarly, Dep_3 and Dep_5 are valid.
- Dep_4 **Validation**: AC2.2 security level ($level_0$) is *not-equal* to the AC2.3 security level ($level_1$). **Result:** An SLO conflict occurs, thus the customer modifies AC2.2 (dependent SLO) to yes ($level_1$) to satisfy the dependency constraint (as shown in Case I "*rev*" column in Table 2.6). In the same way, Dep_7 is validated.
- Dep_6 **Validation**: Since the $(AC2.4, BC2.2, \neq) \in C_v^{\rightarrow_K}$ and AC2.4 security level ($level_1$) is *not-equal* to the BC2.2 security level ($level_0$), $v(AC2.4) \neq v(BC2.2)$ the constraint is satisfied. **Result:** Valid.
- Dep_8 **Validation**: IS1.1 security level ($weekly$ which is $level_2$) is *not-equal* to the IS1.2 security level ($level_3$). **Result:** SLO conflict occurs. The customer changes IS1.1 to $daily$ ($level_3$).

After the customer has resolved all the SLO conflicts and the CSPs secSLAs are validated, each secSLA is mapped to a DSM (cf., Section 2.3.5) to embody the secSLA hierarchical structure. This structure is used in the ranking of CSPs according to the customer requirements (cf., Section 2.3.6). The ranking computation process for Cloud security SLOs defined in Table 2.6 is explained step-by-step using QHP and fuzzy-QHP.

2.4.1.2 *QHP*

For the Audit & Compliance control of Cloud secSLA, there are three security controls (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 2.6, Equations 2.2 is used to define the AC1.2 pairwise relation such that:

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

Thus, the CM of AC1.2 is calculated using Equation 2.5 as:

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

The relative ranking of the CSPs for AC1.2 is given by the priority vector of $CM_{AC1.2}$

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.2 & 0.2 & 0.3 & 0.3 \end{pmatrix} \end{array}$$

This implies that only CSP_3 satisfies the customer requirement for AC1.2. In a similar way, we calculate $CM_{AC1.1}$ and $PV_{AC1.1}$. The AC1 priority vector is then premeditated by aggregating $PV_{AC1.1}$ and $PV_{AC1.2}$ with the normalized dependency levels (which are defined as weights nw_{AC1}) where AC_1 is *medium* dependent on *AC1.1* and *strongly* dependent on *AC1.2* then after normalization:

$$nw_{AC1} = \begin{array}{c} \begin{array}{cc} AC1.1 & AC1.2 \end{array} \\ \begin{pmatrix} \frac{2}{5} & \frac{3}{5} \end{pmatrix} \end{array}$$

Thus, PV_{AC1} is calculated using Equation 2.6 such that:

$$PV_{AC1} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cc} PV_{AC1.1} & PV_{AC1.2} \end{array} \\ \begin{pmatrix} 0.25 & 0.2 \\ 0.25 & 0.2 \\ 0.25 & 0.3 \\ 0.25 & 0.3 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{AC1} \\ \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \end{array}$$

$$PV_{AC1} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.22 & 0.22 & 0.28 & 0.28 \end{pmatrix} \end{array}$$

This means that only CSP_3 satisfies the customer requirements for AC1 as shown in Figure 2.9. Similarly, the Independent Audits and Regulatory Mapping priority vectors are calculated. Subsequently, the three Audit & Compliance services AC1, AC2, AC3 priority vectors are aggregated to have the overall Audit & Compliance priority vector PV_{AC}.
In a similar way, the Business Continuity and Interface Security priority vectors are considered, such that the IS1 priority vector is calculated by aggregating $PV_{IS1.1}$, $PV_{IS1.2}$ and $PV_{AC3.1}$ with the normalized dependency levels (nw_{IS1}) using Equation 2.6.

$$PV_{IS1} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.21 & 0.21 & 0.29 & 0.29 \end{pmatrix} \end{array}$$

This means only CSP_3 satisfies IS1 customer requirement as shown in Figure 2.9. Finally, the priority vectors of Audit & Compliance, Business Continuity and Interface Security are aggregated to obtain the total secSLA priority vector PV_{Root}.

$$PV_{Root} = \begin{array}{c} \begin{array}{ccc} PV_{AC} & PV_{BC} & PV_{IS} \end{array} \\ \begin{array}{c} CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{pmatrix} 0.1693 & 0.2260 & 0.21 \\ 0.2357 & 0.2267 & 0.21 \\ 0.2975 & 0.2736 & 0.29 \\ 0.2975 & 0.2736 & 0.29 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{Root} \\ \begin{pmatrix} 0.3333 \\ 0.3333 \\ 0.3333 \end{pmatrix} \end{array}$$

$$PV_{Root} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.2018 & 0.2241 & 0.2870 & 0.2870 \end{pmatrix} \end{array}$$

Consequently, CSP_3 is the only provider that fulfills the customer's requirements. That was expected, as CSP_1 is not offering AC2.2, AC2.3, AC3.2 and is under-provisioning IS1.1 and IS1.2. CSP_2 is not providing BC2.2 and is not fulfilling customer requirements for AC1.2, AC3.3, BC11.2, IS1.1 and IS1.2. Only CSP_3 fulfills all the customer's requirements. As a result, CSP_3 is the best matching provider according to the customer's requirements, followed by CSP_2 and CSP_1, as shown in Figure 2.11.

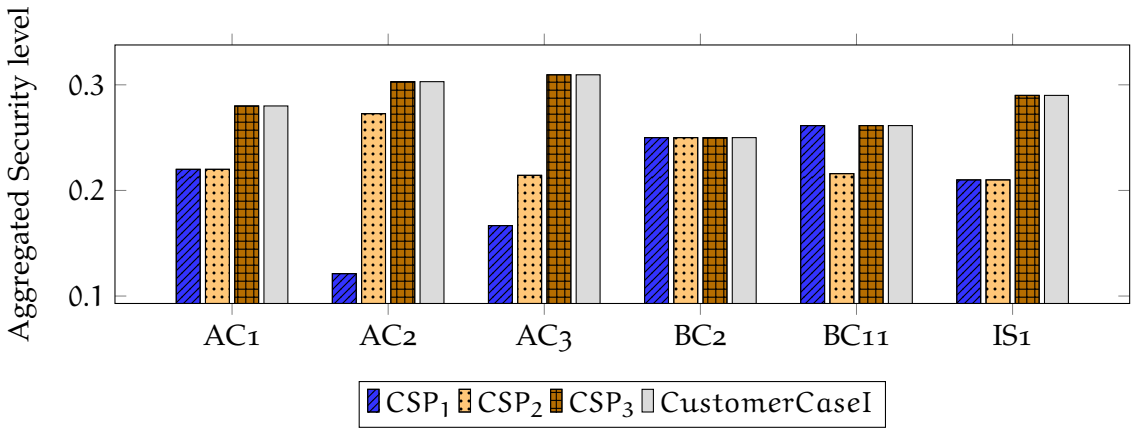

Figure 2.9.: CSPs comparison with respect to Customer Case I requirements using QHP.

2.4.1.3 *Fuzzy-QHP*

The customer specifies his/her requirements at the lowest level of the secSLA (i.e., SLOs) as shown in Table 2.6. In this case, each SLO k_i value is mapped to a progressive TFN value according to its order such that $k_i = \tilde{1} < \tilde{2} < \ldots < \tilde{j}$. These TFN values are represented as $k_i = (1,1,1) < (2,2,2) < \ldots < (j,j,j)$.

For the *Audit & Compliance control* control of Cloud secSLA, *AC1.2* the providers and the customer can specify their values from $level_1$ to $level_3$. Using the data shown in Table 2.6, Equation 2.7 is used to define the *AC1.2* pairwise relation such that:

$$CSP_1/CSP_2 = \frac{\tilde{2}}{\tilde{2}} = (\frac{2}{2},\frac{2}{2},\frac{2}{2}),\ \ CSP_2/CSP_1 = \frac{\tilde{2}}{\tilde{2}} = (1,1,1),\ \ CSC/CSP_1 = \frac{\tilde{3}}{\tilde{2}} = (\frac{3}{2},\frac{3}{2},\frac{3}{2})$$

Thus, the comparison matrix of *AC1.2* $\tilde{A}_{AC1.2}$ as specified in Equation 2.9 is:

$$\tilde{A}_{AC1.2} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} (1,1,1) & (1,1,1) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) \\ (1,1,1) & (1,1,1) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) \\ (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (1,1,1) & (1,1,1) \\ (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (1,1,1) & (1,1,1) \end{pmatrix} \end{array}$$

Then, using Chang's extent analysis method explained in Section 2.3.6.2, we get the relative ranking of the Cloud providers for *AC1.2*, which is given by the priority vector of $\tilde{A}_{AC1.2}$ ($PV_{AC1.2}$). $PV_{AC1.2}$ is calculated as follows. Using Equation 2.10 we get the value of the fuzzy synthetic extent for $\tilde{A}_{AC1.2}$ such that:

$$E_{CSP_1} = (3.33,3.33,3.33)\otimes(\frac{1}{16.67},\frac{1}{16.67},\frac{1}{16.67}) = \begin{array}{c} \begin{array}{ccc} l_{CSP_1} & m_{CSP_1} & u_{CSP_1} \end{array} \\ \begin{pmatrix} 0.2, & 0.2, & 0.2 \end{pmatrix} \end{array}$$

$$E_{CSP_2} = (3.33,3.33,3.33)\otimes(\frac{1}{16.67},\frac{1}{16.67},\frac{1}{16.67}) = \begin{array}{c} \begin{array}{ccc} l_{CSP_2} & m_{CSP_2} & u_{CSP_2} \end{array} \\ \begin{pmatrix} 0.2, & 0.2, & 0.2 \end{pmatrix} \end{array}$$

$$E_{CSP_3} = (5,5,5)\otimes(\frac{1}{16.67},\frac{1}{16.67},\frac{1}{16.67}) = \begin{array}{c} \begin{array}{ccc} l_{CSP_3} & m_{CSP_3} & u_{CSP_3} \end{array} \\ \begin{pmatrix} 0.3, & 0.3, & 0.3 \end{pmatrix} \end{array}$$

$$E_{CSC} = (5,5,5)\otimes(\frac{1}{16.67},\frac{1}{16.67},\frac{1}{16.67}) = \begin{array}{c} \begin{array}{ccc} l_{CSC} & m_{CSC} & u_{CSC} \end{array} \\ \begin{pmatrix} 0.3, & 0.3, & 0.3 \end{pmatrix} \end{array}$$

Afterwards, using Equation 2.11 we get the degree of possibility so that:
$D(E_{CSP_1} \geqslant E_{CSP_2}) = 1$ (as $m_{CSP_1} \geqslant m_{CSP_2}$),
$D(E_{CSP_1} \geqslant E_{CSP_3}) = 0$ (as $l_{CSP_3} \geqslant u_{CSP_1}$),
$D(E_{CSP_1} \geqslant E_{CSC}) = 0$ (as $l_{CSC} \geqslant u_{CSP_1}$)

Then, the possibility for a fuzzy number to be greater than other fuzzy numbers is calculated using Equation 2.12:

$$d'(A_{CSP_1}) = \min(D(E_{CSP_1} \geqslant E_{CSP_2}, E_{CSP_3}, E_{CSC})) = \min(1, 0, 0) = 0$$

Similarly $d'(A_{CSP_2}), d'(A_{CSP_3})$, and $d'(A_{CSC})$ are calculated using Equations 2.11 and 2.12. Thus, the *AC1.2* priority vector PV is given by:

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

This reflects which of the CSPs provide the *AC1.2* security SLO relative to other CSPs and to the customer requirements. After normalization, $PV_{AC1.2}$ is:

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0 & 0 & 0.5 & 0.5 \end{pmatrix} \end{array}$$

This means that only CSP_3 equally satisfies CSC's *AC1.2* requirement. However, CSP_1 and CSP_2 do not fulfill that requirement. Similarly, the priority vector of *AC1.1* is calculated using its comparison matrix $\tilde{A}_{AC1.1}$. The *AC1* priority vector is then premeditated, using Equation 2.14, by aggregating $PV_{AC1.1}$ and $PV_{AC1.2}$ with the dependency relation importance level (nw_{AC1}) defined in Column "lvl" in Table 2.6. Thus,

$$PV_{AC1} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.1 & 0.1 & 0.4 & 0.4 \end{pmatrix} \end{array}$$

Independent audits and *Regulatory mapping* priority vectors are calculated the same way such that:

$PV_{AC2} =$

$$\begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cccccc} PV_{AC2.1} & PV_{AC2.2} & PV_{AC2.3} & PV_{AC2.4} & PV_{AC3.1} & PV_{AC3.2} \end{array} \\ \begin{pmatrix} 0.25 & 0 & 0 & 0.333 & 0.25 & 0 \\ 0.25 & 0.333 & 0.333 & 0 & 0.25 & 0.333 \\ 0.25 & 0.333 & 0.333 & 0.333 & 0.25 & 0.333 \\ 0.25 & 0.333 & 0.333 & 0.333 & 0.25 & 0.333 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{AC2} \\ \begin{pmatrix} 0.27 \\ 0.27 \\ 0.18 \\ 0.09 \\ 0.09 \\ 0.09 \end{pmatrix} \end{array}$$

Therefore,

$$PV_{AC2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.12 & 0.28 & 0.3 & 0.3 \end{pmatrix} \end{array}$$

$$PV_{AC3} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{ccc} PV_{AC3.1} & PV_{AC3.2} & PV_{AC3.3} \end{array} \\ \begin{pmatrix} 0.25 & 0 & 0 \\ 0.25 & 0.333 & 0 \\ 0.25 & 0.333 & 0.5 \\ 0.25 & 0.333 & 0.5 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{AC3} \\ \begin{pmatrix} 0.285 \\ 0.285 \\ 0.43 \end{pmatrix} \end{array}$$

Thus,

$$PV_{AC3} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.071 & 0.166 & 0.38 & 0.38 \end{pmatrix} \end{array}$$

Subsequently, the three *Audit & Compliance* controls *AC1*, *AC2*, *AC3* priority vectors are aggregated to have the overall *Audit & Compliance* priority vector PV_{AC}.In a similar way, the *Business Continuity* and *Interface Security* priority vectors are considered. The CSPs secSLA's rankings according to the customer requirements at the control category level are shown in Figure 2.10. Finally, the priority vectors of *Audit & Compliance*, *Interface Security*, and *Interface Security* are aggregated to obtain the total secSLA priority vector, as shown in Figure 2.11.

$$PV_{total} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{ccc} PV_{AC} & PV_{BC} & PV_{IS} \end{array} \\ \begin{pmatrix} 0.1 & 0.18 & 0.0625 \\ 0.18 & 0.16 & 0.0625 \\ 0.36 & 0.33 & 0.4375 \\ 0.36 & 0.33 & 0.4375 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{total} \\ \begin{pmatrix} 0.3333 \\ 0.3333 \\ 0.3333 \end{pmatrix} \end{array}$$

Thus,

$$PV_{total} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.114 & 0.134 & 0.375 & 0.375 \end{pmatrix} \end{array}$$

Consequently, CSP_3 is the only provider who fulfills the customer's requirements, as shown in Figure 2.11. That was expected, as CSP_1 is not offering *AC2.2*, *AC2.3*, *AC3.2* and is under-provisioning *AC1.2* and *AC3.3*. CSP_2 is not providing *BC2.2* and is not fulfilling customer requirements for *AC1.2*, *AC3.3*, *IS1.1* and *IS1.2*. Similarly, CSP_2 is not fulfilling the customer requirements. Only CSP_3 fulfills customer's requirements and, as a result, CSP_3 is the best matching provider according to the customer's requirements, followed by CSP_2 then CSP_1, as shown in Figure 2.11.

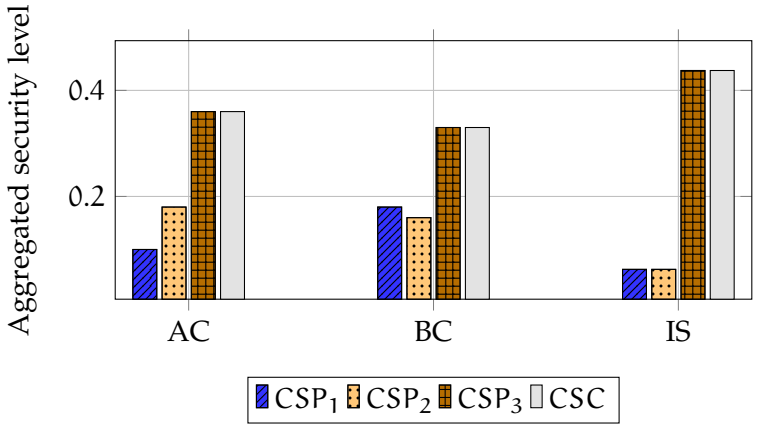

Figure 2.10.: CSPs comparison with respect to customer Case I requirements using fuzzy-QHP.

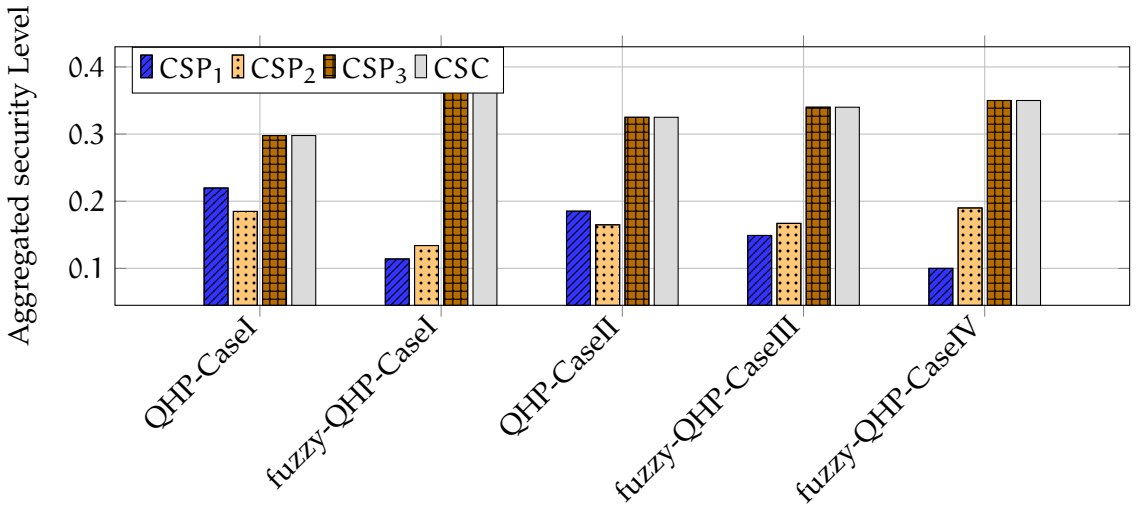

Figure 2.11.: The total aggregated secSLA level with respect to customer requirements using QHP & fuzzy-QHP.

2.4.1.4 *Fuzzy-QHP and QHP comparison*

Although, QHP allows customers to specify their priorities at varied levels of granularity, it only allows customers to provide defined values specifying their requirements and submit static weights to model their priorities (i.e., only "Case I" can be used by QHP; as QHP could not model customer uncertain requirements shown in "Case III" and "Case IV"). As depicted in Figure 2.11, fuzzy-QHP and QHP show the same rankings for customer "Case I" requirements, but with different scales. This indicates that fuzzy-QHP provides the same rankings if the customer specifies the SLO level requirements. In addition to, only fuzzy-QHP methodology is capable of modelling the customer uncertain requirements as explained in the following sections. This denotes that fuzzy-QHP can model customers uncertain requirements and give the CSPs fulfilling customer requirements rankings. This is done without assuming fixed numerical values to each SLO metric (the QHP case), which creates an unrealistic scale of judgment and evaluation.

2.4.1.5 *Cloud Customer Case II Requirements - SLO level after structuring using DSM*

In this case we consider a novice customer who cannot specify his/her precise SLO requirements and/or resolve the SLOs conflicts. The structured secSLA presented using DSM enables the customer to easily specify his/her requirements, regardless of the size of the secSLA and the number of dependencies. In this case the customer defines the least dependent SLOs - these are specified using DSM and are shown in the column marked as Case II in Table 2.6. We explain this Case using only QHP as similarly will be fuzzy-QHP.

Using the data shown in Table 2.6, Equations 2.1 defines the AC1.1 pairwise relation as in Case I. Then the relative ranking of the CSPs for AC1.1 is given by the priority vector calculated using Equation 2.13 (as explained in Case I).

$$PV_{AC1.1} = \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

Similarly, all the lowest level SLOs (least dependent SLOs) are calculated. The $PV_{AC3.1}$, $PV_{AC3.2}$, $PV_{AC3.3}$, $PV_{BC2.2}$, $PV_{BC11.1}$, $PV_{BC11.2}$, $PV_{IS1.1}$ and $PV_{IS1.2}$ are calculated in a

similar way. Then based on the DSM order AC1.2 is calculated. AC1.2 is depending on IS1.2 (Dep_1), thus the $PV_{AC1.2}$ is equal to $PV_{IS1.2}$.

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.2 & 0.2 & 0.3 & 0.3 \end{pmatrix} \end{array}$$

In the same way, $PV_{AC2.3}$, $PV_{AC2.4}$ and $PV_{BC2.1}$ are calculated (they are equal to $PV_{AC3.2}$, $PV_{BC2.2}$ and $PV_{BC11.1}$ respectively). Furthermore, AC2.1 is calculated. AC2.1 depends on AC1.1 and AC3.1 (Dep_1 and Dep_3) with different levels of dependencies. Thus, using Equation 2.6:

$$PV_{AC2.1} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cc} PV_{AC1.1} & PV_{AC3.1} \end{array} \\ \begin{pmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \\ 0.25 & 0.25 \end{pmatrix} \end{array} \begin{array}{c} nw_{AC2.1} \\ \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \end{array}$$

After all the SLO priority vectors are determined, the priority vectors are aggregated with the dependency importance level to get the overall rank of CSPs according to the customer requirements as specified in Case I (as depicted in Figure 2.11). As a result the root priority vector is equal to:

$$PV_{Root} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.2018 & 0.2241 & 0.2870 & 0.2870 \end{pmatrix} \end{array}$$

2.4.1.6 *Cloud customer Case III Requirements - different levels of granularity*

The customer specifies his/her requirements using linguistic descriptors (depicted in Figure 2.8) at different levels of the secSLA hierarchical structure as shown in column "Case III" in Table 2.6. We assume that the customer specifies *Highly-Required* to *Audit planning* and *Independent audits* as well as *Not-Required* to *Regulatory mapping*. Furthermore, we assume the customer specifies *Do-not-know* for *Security architecture*, and specifies low-level requirements for *Resiliency*, as shown in Table 2.6. Moreover, the customer submit qualitative labels to specify the required level of importance.
Since *Audit planning* is assigned *HR*, the respective SLOs value of both *AC1.1* and *AC1.2* are set to $(2,3,4)$. Note that, we model the pair wise relation of the *HR* customer requirements ($\tilde{3}$) as CSC/CSC which is equal to (1,1,1) and not $(\frac{2}{4},\frac{3}{3},\frac{4}{2})$. Thus, for *AC1.2* the pairwise relation is defined using Equation 2.7 such that:

$$CSP_3/CSC = (\frac{3}{4},\frac{3}{3},\frac{3}{2}),\ \ CSC/CSP_1 = (\frac{2}{2},\frac{3}{2},\frac{4}{2}),\ \ \mathsf{CSC/CSC} = (1,1,1)$$

Thus, the comparison matrix of *AC1.2* $\tilde{A}_{AC1.2}$ as specified in Equation 2.9 is:

$$\tilde{A}_{AC1.2} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} (1,1,1) & (1,1,1) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) & (\frac{2}{4},\frac{2}{3},\frac{2}{2}) \\ (1,1,1) & (1,1,1) & (\frac{2}{3},\frac{2}{3},\frac{2}{3}) & (\frac{2}{4},\frac{2}{3},\frac{2}{2}) \\ (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (\frac{3}{2},\frac{3}{2},\frac{3}{2}) & (1,1,1) & (\frac{3}{4},\frac{3}{3},\frac{3}{2}) \\ (\frac{2}{2},\frac{3}{2},\frac{4}{2}) & (\frac{2}{2},\frac{3}{2},\frac{4}{2}) & (\frac{2}{3},\frac{3}{3},\frac{4}{3}) & (1,1,1) \end{pmatrix} \end{array}$$

Afterwards, using Equations 2.10, 2.11 and 2.12 the *AC1.2* priority vector PV is given by:

$$PV_{AC1.2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.115 & 0.115 & 0.38 & 0.38 \end{pmatrix} \end{array}$$

PV_{AC1} is then calculated by aggregating $PV_{AC1.1}$ and $PV_{AC1.2}$ such that:

$$PV_{AC1} = \begin{array}{c} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{array} \begin{array}{c} \begin{array}{cc} PV_{AC1.1} & PV_{AC1.2} \end{array} \\ \begin{pmatrix} 0.25 & 0.115 \\ 0.25 & 0.115 \\ 0.25 & 0.38 \\ 0.25 & 0.38 \end{pmatrix} \end{array} \cdot \begin{array}{c} nw_{AC1} \\ \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \end{array}$$

Thus,

$$PV_{AC1} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.169 & 0.169 & 0.328 & 0.328 \end{pmatrix} \end{array}$$

Similarly the *Independent audits* SLOs value are set to $(2,3,4)$. On the other hand, *Regulatory mapping* is denoted as *NR* by the customer, which means it will not affect the overall security level of the providers as it is not required by the customer. Therefore, PV_{AC1}, PV_{AC2} and PV_{AC3} are aggregated with nw_{AC} such that:

$$PV_{AC2} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.12 & 0.268 & 0.305 & 0.305 \end{pmatrix} \end{array}$$

Therefore PV_{AC} is equal to:

$$PV_{AC} = \begin{array}{c} \begin{array}{cccc} CSP_1 & CSP_2 & CSP_3 & CSC \end{array} \\ \begin{pmatrix} 0.1445 & 0.2185 & 0.3165 & 0.3165 \end{pmatrix} \end{array}$$

This implies that CSP_1 and CSP_2 do not fulfill CSC *Audit & Compliance* control and only CSP_3 satisfies *AC* requirement. For *Business Continuity*, the customer specifies *Do-not-know*, which is assigned as $(1, 2.5, 4)$. Thus,

$$PV_{BC} = \begin{matrix} \\ \end{matrix} \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \\ \begin{pmatrix} 0.24 & 0.22 & 0.27 & 0.27 \end{pmatrix}$$

Similarly, *Interface Security* is evaluated as explained in Case I. Next, PV_{AC}, PV_{BC}, and PV_{IS} are aggregated to obtain the total secSLA priority vector, as shown in Figure 2.11.

$$PV_{total} = \begin{matrix} \\ \end{matrix} \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \\ \begin{pmatrix} 0.149 & 0.167 & 0.34 & 0.34 \end{pmatrix}$$

Therefore, only CSP_3 satisfies the customer needs, whereas both CSP_1 and CSP_2 do not fulfill customer requirements, as shown in Figure 2.11. That was expected, as *AC2.3* is highly required by the customer and not provided by CSP_1. Moreover, CSP_1 and CSP_2 are not fulfilling the customer IS requirement. The presented framework can give accurate CSP ranking even if the low level is not defined and vague preferences are specified at the highest levels, which means a customer can define weights at the higher levels instead of answering multiple low-level questions.

2.4.1.7 *Cloud Case IV requirements - natural language sentence*

The customer requires "a provider with only extremely high *Audit & Compliance* and does not require *Business Continuity* and *Interface Security*". Using Rule 1, the customer allocates *Highly-Required* for *Audit & Compliance*, and *Not-Required* for *Security architecture* and *Resilience*. Similarly, as shown in previous cases, the priority vector of *AC* is calculated, such that:

$$PV_{AC} = \begin{matrix} \\ CSP_1 \\ CSP_2 \\ CSP_3 \\ CSC \end{matrix} \begin{matrix} PV_{AC1} & PV_{AC2} & PV_{AC3} \end{matrix} \\ \begin{pmatrix} 0.1 & 0.12 & 0.08 \\ 0.1 & 0.268 & 0.19 \\ 0.4 & 0.305 & 0.36 \\ 0.4 & 0.305 & 0.36 \end{pmatrix} \cdot \begin{matrix} nw_{AC} \\ \end{matrix} \\ \begin{pmatrix} 0.333 \\ 0.333 \\ 0.333 \end{pmatrix}$$

Therefore,

$$PV_{AC} = \begin{matrix} \\ \end{matrix} \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \\ \begin{pmatrix} 0.1 & 0.19 & 0.35 & 0.35 \end{pmatrix}$$

As *BC* and *IS* are not required by the user, thus:

$$PV_{total} = \begin{matrix} \\ \end{matrix} \begin{matrix} CSP_1 & CSP_2 & CSP_3 & CSC \end{matrix} \\ \begin{pmatrix} 0.1 & 0.19 & 0.35 & 0.35 \end{pmatrix}$$

Consequently, only CSP_3 is satisfying the customer requirements, followed by CSP_2 then CSP_1. Therefore, the presented framework can give accurate CSPs ranking even if the customer only specified requirements using natural language sentences.

2.4.2 *The CSP Perspective: Maximising Offered Security Levels*

The second validation scenario presented applies the secSLA evaluation techniques to solve problems faced by CSPs i.e., which specific security SLO from the offered secSLA should be improved in order to maximise the overall security level according to the customer requirements? This might be the case of a well-established CSP deciding where to invest in order to achieve the highest possible security level, or a new CSP designing the secSLA. To answer this question, we could perform a sensitivity analysis to ascertain the security benefits of improving one or more SLOs. However, this analysis becomes impractical as the number of SLOs and the dependencies between them increase. Thus the sensitivity analysis is performed on the least dependent SLOs identified by the DSM.

We used the CSP_1 dataset described at Table 2.6, and applied the Case II requirements to setup the customer's baseline for the security evaluation. From the existing 9 least dependent SLOs (Case II column in Table 2.6) the CSP_1 is under-provisioning 4 of them (AC3.2, AC3.3, IS1.1 and IS1.2). Figure 2.12 shows how the proposed framework can be used to analyse an existing secSLA, and extract the individual SLOs that, if enhanced, would result on different improvements associated to the overall security level. In this case, the X-axis represents the improvement associated to the overall security level after enhancing any of the SLOs. It is shown as a percentage where 0% corresponds to the original secSLA and 100% is the most effective SLO. For example, providing tenants with the security policies applicable to virtualised resources (AC3.2 in Figure 2.12), quantitatively increases CSP_1 security level better than improving the thresholds committed for any of the other SLOs.

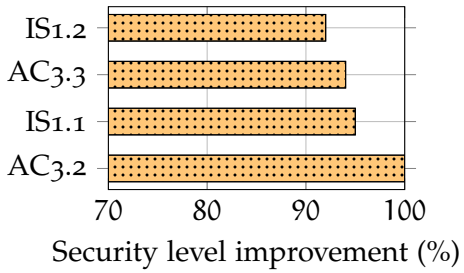

Figure 2.12.: Sensitivity analysis: CSP_1 SLOs that maximise the overall security level.

2.5 RELATED WORK

Multiple approaches have been emerging to assess the functionality and security of CSPs. In [LYKZ10], Li et al. proposed a framework to compare different Cloud providers across performance indicators. Garg et al. [GVB13] proposed an Analytic Hierarchy Process (AHP) based ranking technique that utilizes performance data to measure various Quality of Service (QoS) attributes and evaluates the relative ranking of CSPs. A framework of critical characteristics and measures that enable comparison of Cloud services is presented by Siegel et al. [SP12]. However, these studies focused on assessing performance of Cloud services and not their security properties.

While some approaches have focused on specifying Cloud security parameters in secSLAs, fewer efforts exist for quantifying these SLA security services. Henning et al. [Hen99] identified security SLAs with applicable types of quantifiable security metrics for non Cloud systems. These metrics were expanded by Irvine et al. [IL01] outlining the

term QoSS for quality of security service. Based on QoSS, Lindskog et al. [Lin05] defined four dimensions that specify a tunable Cloud security service.

Security requirements for non Cloud services have been addressed by Casola et al. [CMMR06], who proposed a methodology to evaluate security SLAs for web services. Chaves et al. [CWL10] explored security in SLAs by proposing a monitoring and controlling architecture for web services. As pointed out by Chaves et al., it is a challenge to define quantifiable security metrics, but they give examples related to password management, frequency of backups and repair/recovery time. A technique to aggregate security metrics from a web services' secSLAs has been proposed by Frankova et al. [FY07] and krautsevich et al. [KMY11]. However, the authors did not propose any techniques to assess Cloud secSLAs or empirically validate the proposed metrics. In [AGI11], Almorsy et al. propose the notion of evaluating Cloud secSLAs by introducing a metric to benchmark the security of a CSP based on categories. However, the resulting security categorization is purely qualitative and lacks the support of dependencies. Luna et al. [LGLS12] presented a methodology to quantitatively benchmark Cloud security with respect to Customer defined requirements (based on control frameworks). However, the presented techniques only considered SLO values assignment at the SLO level. Mostly, they depend on a static weight assessment for preference modeling, which (a) is time consuming, (b) does not take into account the uncertainty associated with the mapping of customers' judgement to a number, as human assessment of qualitative attributes is always subjective and thus imprecise, and (c) did not consider the dependencies across services and conflicts detection.

To address customers uncertainty, fuzzy logic has been applied by Wang et al. [Wan09]. The authors used a fuzzy-based MCDM algorithm to rank web services. Hamzeh et al. [AM13] developed a fuzzy-based MCDM approach that uses linguistic descriptors to model preferences. However, all these studies focus on the performance of Cloud services and not on security properties. Noor et al. [NS11] and Habib et al. [HRM11] evaluated trust of Clouds according to customer feedback. Nevertheless, they ignored customers' diverse requirements. In [Sup+12], Supriya et al. used a fuzzy inference system to evaluate the trust of providers. However, they required that customers manually tune the inference system according to their expectations for each query. Furthermore, the dependencies and conflict detection are not covered in any of them.

Dunlop et al. [DIR02] proposed a model to specify policies of permission, prohibition and obligation in a temporal logic language that can reason about the sequences of events. In [CYZ+12], Chen et al. presented a framework for automatic detection of conflicts covering violation of enterprise policies and inconsistency of customer requirements. Ensel et al. [EK01] introduced an approach to handle dependencies between managed resources (e.g., web application server, database) in a distributed system. However, the support for secSLA management is not provided. The COSMA approach [LF08] supports the providers of composite services to manage their SLAs. However, COSMA does not support the determination of the effect of SLO violations on other services based on dependency information. Also non of the specified techniques considered services' dependencies.

2.6 SUMMARY

In this chapter, we developed two techniques to conduct security assessment based on the quantitative and qualitative analysis of security information. We propose techniques to conduct quantification of information, techniques for information composition and for

security comparisons across Cloud providers. The proposed techniques were designed based on the specifics of secSLAs from state-of-the-art works and standardization bodies. Furthermore, both techniques were empirically validated through case studies using real-world CSP data obtained from the Cloud Security Alliance. The validation experiments were useful not only to highlight the applicability of our techniques to real-world CSPs but also to highlight the advantages and limitations of these techniques, and to provide an objective comparison of both of them in order to guide (prospective) adopters. Furthermore, we developed a novel dependency model and evaluated its capability for validating each customer and CSP requirements.

In the next chapter, we utilize the customer defined requirements as a reference level of details to drive requirement based threat analysis. The proposed requirement based threat analysis is an exploration to determine the potential of ascertaining design-level threats from the analysis of requirements. The completeness of a requirement based threat analysis inherently depends on the level of detail of the provided requirements.

Part II

THREAT ANALYSIS

3

REQUIREMENT BASED THREAT ANALYSIS

In the preceding chapter we presented a security assessment and validation of Cloud providers according to the customer's security requirements. On the one hand, the assessment process includes two different evaluation techniques as specified in the previous chapter. On the other hand, the validation process captures information about secSLA services' dependencies and checks service conflicts and different services compatibility issues. In this chapter, we explore threat analysis at the requirements level while taking service dependencies into consideration, named Requirement Based Threat Analysis (RBTA). Specifically, the threat analysis focuses on threats that can potentially violate the customer's data ownership requirements of security, functionality and performance. Thus, we present a systematic analytic RBTA process which is able to establish the viability of identifying threats based on a customer requirements analysis. The requirements also form the basis for testing and validation.

3.1 MOTIVATION AND CONTRIBUTION

Typical dependency relations across requirements can easily introduce conflicts leading to various threats. Consequently, threat analysis process is used to establish the viability of identifying threats based on customer requirements. However, generally this process is manual which is both time intensive and makes no assurance on completeness of the analysis

Accordingly, we develop RBTA approach to systematically conduct the process of ascertaining and visualizing the dependencies across the requirements to identify threats. Specifically, RBTA approach provides an input for estimating the risks based on such requirement analysis. Thus, building upon the basic viability of the RBTA result, the proposed approach developed a systematic analytical technique that enumerates a set of customer use-case (UC) requirements and then determines all possible direct/indirect dependencies across them to conduct a generalized threat analysis from their requirements. The initial UC specific RBTA approach can be extended to apply to generic UC via the development of a generalized Cloud threat model.

Contributions. We make the following contributions:

1. Developing an efficient architectural threat modeling approach that allows partly automating and focusing security assessment based on assets and security objectives in the system. The threat analysis involves developing a dependency model and showing its capability for identifying threats by visualizing the dependencies across services. The threat analysis utilizes information from the customer requirements, the number of requirements, the maximum/average values for the *Violation Likelihood* and *Threat Severity*, and the ordering of the requirements from the least to the most dependent services. These *Violation Likelihood* and *Threat Severity* measurements as well as the violations identification are considered to illustrate the violations severity that could potentially occur across the services.

2. Validating the presented approach, for its effectiveness, on a ESCUDO-CLOUD European project UC [Pro17]. The approach can be also generalized to apply to other requirements. Using the specified UC's security-related information, an automated threat identification is achieved.

3.2 REQUIREMENT BASED THREAT ANALYSIS (RBTA) PROCESS

Threat analysis (TA) is a process to systematically identify, detect, and evaluate security vulnerabilities. Threat analysis considers the full spectrum of threats for a given system. For example, TA can be conducted at the level of architecture diagrams, at trust boundaries, over detailed data flow diagrams, attack surfaces at the component or functional levels, on source code, via misuse-cases and many other dimensions.

We explore TA at the requirements level by considering two main stages namely, requirements analysis and dependency analysis. These two main stages are conducted in progressive steps:

Step (1) Identifying security requirements from multiple dimensions. These dimensions are a) security properties (Confidentiality, Integrity and Availability); b) sharing requirements; c) access requirements. For example, CRUD (Create, Read, Update, and Delete) operations for infrastructure and tenants' keys and scalable design of key management system.

Step (2) Identifying the relationship between requirements and their impact on performance, functionality and security properties.

Step (3) Assigning a priority level to each requirement to ascertain the dependencies among the requirements, e.g., scalable design of key management depends on the CRUD operations for infrastructure and tenants' keys. This helps to better understand the threats associated to each requirement and the consequence of its violation on other requirement(s).

Step (4) Once we have identified the basic requirements and their dependencies, the next step is to ascertain the assumptions behind the requirements, which can be direct and indirect. For example, the direct assumption for CRUD operations for tenant keys is the need of a trusted administrator and the indirect assumption is the need for a trusted infrastructure. If such assumptions are violated, it will result in confidentiality violation for the tenant data which is a high severity threat.

Step (5) The final step is to model security requirements and their dependencies utilizing a hierarchical/tree structure.

3.3 REQUIREMENTS ANALYSIS

In this section we consider various Cloud requirements (i.e., services) using ESCUDO-CLOUD use-case [Pro17].

3.3.1 *Requirements Analysis from ESCUDO Use-Case*

Cloud computing requires data owners to outsource their data to a Cloud provider. To prevent unauthorized access, data has to be encrypted before outsourcing. However, the

data still has to be utilizable for business purposes. Most business scenarios, like the ones involving complex supply chains, involve several parties which do not necessarily trust each other. This depicts a conflict: on the one hand, sharing of data is a must in order to guarantee a better success of the business. On the other hand, substantial threats have to be taken care of in data sharing. Therefore, one goal of this use-case is to come up with solutions allowing a data owner to share information without losing control over his data. This is realized by developing new systems for cooperation based on encrypted database technology. With such systems data owners should be empowered to outsource their data to the Cloud, while preserving security and functionality. Technology based on the search and aggregation of encrypted data ensures these functionalities. The technology also offers the data owner to selectively grant and revoke data access to their trusted business partners.

3.3.2 *Adapted Requirements Catalogue*

Table 3.1 identifies the subset of requirements relevant for data ownership attributes. Table 3.1 first names and describes use-case (UC) requirements followed by their priority levels and the dependencies across these requirements. Subsequently, the specific attribute of CIA/Performance/Functionality relevant to the requirement is identified. This table now forms the basis of identifying the threats for these requirements.

3.3.3 *Risk Assessment*

Using the relevant data-ownership requirements in Table 3.1, a requisite risk assessment is enumerated in Table 3.2. For each requirement, firstly, we identify the direct assumptions required for the requirement to hold. This is followed by identification of the indirect assumptions needed for the requirement to be supported. Then the violations that could occur between services and their expected (assumed) impact is specified. This is achieved by defining and analyzing the type of risks causing these violations along with their violation likelihood and their severity.

Requirements Reference	Requirement Description	Priority	Dependencies	CIA	Performance	Functionality
REQ-UC-AC1	Access Control per Client	High	REQ-UC-KM1	C		✓
REQ-UC-AC2	Access control per group of clients	High	REQ-UC-KM2	C		✓
REQ-UC-AC3	Access control per database cell	High	REQ-UC-EQ2	C		✓
REQ-UC-AC4	Access control matrix model	Medium		C		✓
REQ-UC-AC5	Access grant and revoke by administrator	Low	REQ-UC-EQ3	CI		✓
REQ-UC-AC6	Access control enforced by client	High	REQ-UC-KM3	CI		✓
REQ-UC-KM1	One key per client	High		CI		✓
REQ-UC-KM2	Group key management	High		CI		✓
REQ-UC-KM3	Client key securely stored at client only	High		CI		✓
REQ-UC-KM4	Group keys derivable	High	REQ-UC-KM2	CI		✓
REQ-UC-EQ1	Encryption schemes	High		C	✓	
REQ-UC-EQ2	Adjustable onion encryption	High	REQ-UC-EQ1	C	✓	
REQ-UC-EQ3	Proxy re-encryption, Query rewriting, Post-processing	High		C	✓	

Table 3.1.: Relationship between use-case requirements and confidentiality, integrity, availability, performance and functionality attributes with regards to data ownership.

Requirement Reference	Requirement Description	Direct Assumptions	Indirect Assumptions	Violation Likelihood (1/Low-10/High)	Assumed Impact	Threat Severity (1/Low-10/High)
REQ-UC-AC1	Access Control per Client	User Authentication		5	Basic assumption violated	8
REQ-UC-AC2	Access control per group of clients	User Groups exist		3	User inconvenience	1
REQ-UC-AC3	Access control per database cell	Cells are controllable		3	Exposure of larger structures, e.g. tables or columns, potentially resulting in partial confidentiality violation	4
REQ-UC-AC4	Access control matrix model	Independent of time and workflow		4	User inconvenience	1
REQ-UC-AC5	Access grant and revoke by administrator	Administrator for group maintenance available		3	User inconvenience	1
REQ-UC-AC6	Access control enforced by client	Trusted Client		5	Confidentiality violation for user data	5

REQ-UC-KM1	One key per client	Key length sufficient, keys are renewable	Secure Storage for Key	2	Keys guessable, global confidentiality violation	10
REQ-UC-KM2	Group key management	Group can secure key, Key is renewable, key length is sufficient		8	Confidentiality violation for group data	5
REQ-UC-KM3	Client key securely stored at client only	Secure Storage		2	Confidentiality violation for user data	5
REQ-UC-KM4	Group keys derivable	Cryptographic assumptions hold		3	Disaster, global confidentiality violation	10
REQ-UC-EQ1	Encryption schemes	Cryptographic assumptions hold	Key is secret (i.e. KM1)	3	Disaster, global confidentiality violation	10
REQ-UC-EQ2	Adjustable onion encryption	Set of supported queries is sufficient, Scalability is given		4	Performance problems	2
REQ-UC-EQ3	Proxy re-encryption, Query rewriting, Post-processing	Security enforcement is possible for Multi user environment		3	User has access to data which was supposed to be revoked, resulting in partial confidentiality violation	4

Table 3.2.: Use-case requirements and their direct and indirect assumptions as well as an estimated likelihood of direct assumptions being violated, assumed impact and threat severity.

3.4 DEPENDENCY ANALYSIS ACROSS REQUIREMENTS

In the preceding section, the use-case requirements and risk assessment table of a UC was specified to establish the viability of a requirements based threat analysis. However, this process is manual and also provides no assurance on completeness. Consequently, the intent in this section is to develop techniques to systematically conduct the process of ascertaining and visualizing the dependencies across the UC requirements to identify threats.

Due to these dependency relations, a security breach causing a violation of one of these services can lead to the violation of other services. Depending on the nature of the incident, these violations can be anything from low-risk to highly critical. Consequently, an explicit knowledge about dependencies between requirements and their threat severity is needed to support the management of Cloud services by both entities (the Cloud provider and/or the customer).

The dependency model specified in Chapter 2 is utilized in this chapter. The dependency management and requirements structuring is performed in progressive stages, as shown in Figure 3.1.

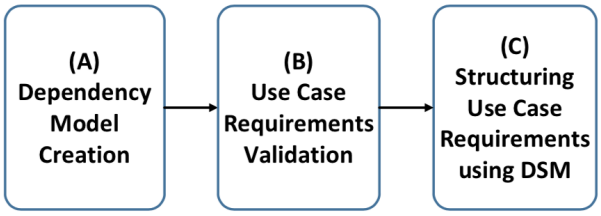

Figure 3.1.: Dependency management approach stages

A dependency model is created in Stage (A) to capture information about the use-case requirements (i.e., Services) and the subsequent dependencies. Subsequently in Stage (B), the dependency model is specified using a machine readable format to allow automated validation for checking service conflicts and different services compatibility issues. Finally in Stage (C), the validated services are structured using the Design Structure Matrix (DSM) [Ste81] depicting dependencies between services as an ordered list.

Based on the analysis of the UC requirements specified before, these requirements are modelled using a graphical mapping in a hierarchical structure, as shown in Figure 3.2. This figure depicts UC requirements as specified in Table 3.1 respectively. We detail each stage in the following sections:

3.4.1 *Stage (A): Dependency Model Creation*

A dependency model is created to cover all identified dependencies within the requirements. This model is based on the dependency model introduced in Section 2.3.4.

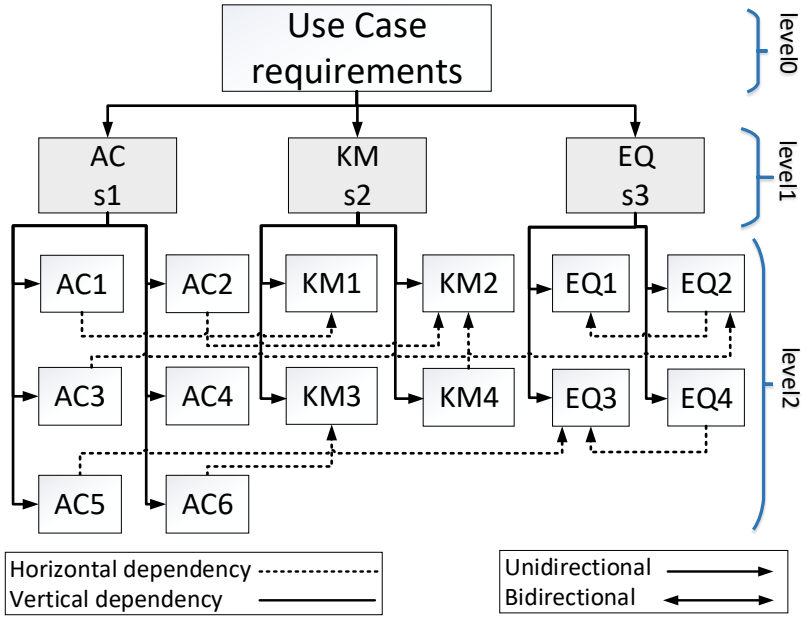

Figure 3.2.: Use-case requirements arranged in a hierarchy structure showing the dependencies between requirements

3.4.2 *Stage (B): Use-Case Requirements Validation*

A meta-model is developed for the UC requirements (cf., Section 2.3.4) based on the dependency definitions. This meta-model allows the description of services along with the information on the UC requirements drafted for it.

3.4.3 *Stage (C): Structuring Use-Case Requirements Using DSM*

The ordering of the use-case requirements is performed using DSM. To demonstrate the idea of DSM, the mapping of the use-case shown in Figure 3.2 into a DSM is depicted in Table 3.3. As the dependencies of services on themselves are not considered (as specified earlier in the dependency model constrains), there are no marks along the diagonal. For example, by examining:

- Row 3 in the DSM mapping of UC requirements (Table 3.3) shows that KM depends on KM1, KM2, KM3, and KM4.

After mapping the use-case requirements into a DSM, we can start reordering the DSM rows and columns in order to transform the DSM into a lower triangular form.

Table 3.4 show the result of partitioning the DSM depicted in Table 3.3 respectively. Bidirectional dependencies occur when the matrix cannot be reordered to have all matrix elements sub-diagonal.

Consequently, Table 3.4 outline all possible direct and indirect dependencies across requirements. This dependency structuring enables the customer to identify, quantify, and address the security risks associated with his/her requirements .

#		UC	AC	KM	EQ	AC1	AC2	AC3	AC4	AC5	AC6	KM1	KM2	KM3	KM4	EQ1	EQ2	EQ3	EQ4
1	**UC**	.	X	X	X														
2	**AC**		.			X	X	X	X	X	X								
3	**KM**			.								X	X	X	X				
4	**EQ**				.											X	X	X	X
5	**AC1**					.						X							
6	**AC2**						.						X						
7	**AC3**							.									X		
8	**AC4**								.										
9	**AC5**									.								X	
10	**AC6**										.			X					
11	**KM1**											.							
12	**KM2**												.						
13	**KM3**													.					
14	**KM4**												X		.				
15	**EQ1**															.			
16	**EQ2**															X	.		
17	**EQ3**																	.	
18	**EQ4**																	X	.

Table 3.3.: DSM mapping of UC requirements shown in Figure 3.2

#		AC4	KM1	KM2	KM3	EQ1	EQ3	AC1	AC2	KM4	AC6	AC5	EQ2	EQ4	AC3	KM	EQ	AC	UC
1	AC4	.																	
2	KM1		.																
3	KM2			.															
4	KM3				.														
5	EQ1					.													
6	EQ3						.												
7	AC1		X					.											
8	AC2			X					.										
9	KM4			X						.									
10	AC6				X						.								
11	AC5						X					.							
12	EQ2					X							.						
13	EQ4						X							.					
14	AC3												X		.				
15	KM		X	X	X					X						.			
16	EQ					X	X						X	X			.		
17	AC	X						X	X		X	X						.	
18	UC															X	X	X	.

Table 3.4.: Final DSM of UC requirements depicted in Table 3.3 after partitioning and scheduling

3.5 IDENTIFYING VIOLATIONS

In order to understand the violations that could occur between the services, we first have to define and measure the risks causing these violations and also their severity. Based on the assumed impact and threat severity of the use-case as specified in Table 3.2, the concept of risk is defined as a measure of the expected negative effect of a particular unwanted event (i.e., Assumed Impact) [JW98]. This is expressed as the product of the *Violation Likelihood* of the event and the expected damage (*Threat Severity*). This measure is then used to determine if these risks are acceptable and to decide which ones to mitigate first.

Table 3.4 outlines all possible direct and indirect dependencies across the requirements. Furthermore, the table enumerates the service requirements as ordered by their criticality. This DSM-based process also ensures completeness of ascertaining the dependencies to the level of information available in the requirements by categorizing the marked X's. Note that, X can be replaced by any numerical number according to the measured risk. In the following, we show an example sampling of the possible use-case violations as identifiable from the developed DSM table corresponding to the UC.

For instance, the breach of *EQ3* leads to the breach of *AC5*, *EQ4*, and *EQ*. For example, this is the case if a user has access to data which was supposed to be revoked, resulting in a partial confidentiality violation. Also, the violation of *KM1* directly affects *AC1* as shown in Table 3.4 column 2. This means that if the keys are compromised and thus one key per client requirement is violated, then the access control per client requirement is violated as well. We also note that by violating the confidentiality of group data, an attacker can obtain access per group of clients, and derive the group keys as depicted in Table 3.4 column 3.

3.6 RELATED WORK

Threat analysis is a well utilized approach to identify software/system threats. However, it is typically constrained to a specific configuration and for a specific attacker profile. The initial efforts led by Microsoft introduced a threat modeling approach called STRIDE [HLOS06]. It is an attacker-centric approach, applicable to data flow diagrams to find potential weaknesses and security flaws exploitable by a specific attacker. The approach presented in [WLK+12] develops an attack tree of the Cloud and utilizes a what if analysis to traverse paths in the tree to determine potential exploit(s). In [MKS13], Malik et al. formally analyzed open source Cloud environments for assessing correctness properties. Perez et al. [PBSL13] characterize vulnerabilities in the hypervisor by considering their impact on the functionality of selected popular hypervisors. The security issues of hypervisors was also analyzed by Tsai et al [TSM+12]. Their analysis focused on security issues over VM hopping and VM mobility. In [BNP+11], Bugiel et al. analyzed publicly available VM images in the Amazon EC2 repository. Their analysis focused on public interfaces to extract private information to launch attacks such as starting a botnet or launching an impersonation attack.

3.7 SUMMARY

Overall, we have been able to: firstly, developing an analytical process that is able to systematically capture both direct and indirect dependencies at the level of the use-case requirements. Secondly, assessing the degree of dependencies across the requirements. Thirdly, validating the effectiveness of a requirements based analytical process. Finally, identifying potential threats that could occur for the ESCUDO-CLOUD UC, and specify the critical areas where services should be protected. The developed process has also been generalized to apply beyond the ESCUDO-CLOUD UC provided that the requirements capture is available.

Part III

QUANTITATIVE SECSLA VALIDATION AND ENFORCEMENT

4

MONITORING THE COMPLIANCE OF CLOUD SERVICES

In this chapter, a framework for secSLA compliance validation is proposed. The proposed framework enables the Cloud customers to validate the compliance of their Cloud SLOs to the contracted security level in the secSLA. This is first achieved by defining methodologies to evaluate the SLOs associated with each security property defined in a secSLA. Then, by monitoring the compliance of the service throughout the service life-cycle, any violations to the contracted values are detected and reported to the customer. The proposed framework builds upon a novel decentralized runtime monitoring approach relying on the Ethereum blockchain infrastructure.

4.1 MOTIVATION AND CONTRIBUTION

Monitoring the service levels described in SLAs is a critical task. The responsibility of this task is usually taken over by either Cloud providers or third-parties.

- **Cloud provider side monitoring:** The CSPs develop their own monitoring tools and market them as part of the Cloud service package (e.g., Amazon CloudWatch [Ama]).
- **Third-party system:** Usually a trusted organization that provides a monitoring tool and takes over the responsibility of monitoring the system's quality levels [MJSCG04]. Examples of popular third-party monitoring tools are Nagios [Nag], and Zabbix [DVL15].

Furthermore, most of the proposed compliance approaches have been proposed for Cloud service performance validation and none of them address compliance management from the customer side.

The problem of establishing security assurance can be solved by proposing mechanisms that enable the customers to: (a) verify if the contracted security level is actually delivered and (b) detect any violations in the secSLA during the service life-cycle. However, each of these elements has its own challenges. Verifying that the security level of the provided service matches the contracted level entails that the customer can assess the security level of the service. Hence, assessing the security level of the service is constrained by finding assessment techniques which allow the customers to evaluate the SLOs; the two security evaluation techniques presented in Chapter 2 address this problem.

To target element (b), the security level of the service should be monitored throughout its full life-cycle, such that any violations to the contracted service is reported. The monitoring mechanism should continuously assess the security level of the service based on the SLOs contained in the secSLA. Monitoring the security level of the service implicates many challenges. First, the monitoring mechanism must be able to identify and detect any violation that may occur to the values of any SLO contained in the secSLA. Further, the monitoring mechanism must prevent CSPs or customers from misreporting for financial gain.

Another problem facing the Cloud customers is the "manual time intensive" compensation process. The customers who experienced security breach, data loss, or outages have

to open a case and report the violation to the CSP's support team. After the case is validated by the CSP's team, a refund is initiated in the form of a future usable credit for customers. To that end, in this chapter a framework for secSLA enforcement and compliance validation as well as autonomous secSLA compensation is proposed. The proposed framework enables the Cloud customer to validate the compliance of their Cloud service to the contracted security level in the secSLA.

The proposed framework builds upon a decentralized runtime monitoring approach relying on the Ethereum blockchain infrastructure. The monitoring approach continuously monitor the compliance of Cloud services to the contracted SLOs. The approach first defines means to evaluate SLOs associated with each security property defined in a secSLA. Our approach relies on the disruptive blockchain technology (i.e., Ethereum blockchain) to revolutionize and automate traditional Cloud secSLAs. Ethereum blockchain is utilized due to its turing-complete smart contract development platform [Woo14] that enabled its wide adoption in developing decentralized apps [Sta]. Our approach digitizes the traditional Cloud secSLA by enforcing secSLA with Ethereum smart contracts. The smart contract aggregates the monitoring logs, proves secSLA violations, and compensates the affected customer according to the severity of the violation.

The approach aims to automate the validation process, eliminating the need for human intervention during the measurement, monitoring and the validation processes.

Contributions. We make the following contributions:

1. Analyzing and selecting the SLOs that can be measured from the customer side. The analysis is performed to understand each SLO and decide upon the possibility of evaluating its value. The analyzed SLOs are proposed by different security controls frameworks and standards.

2. Defining a measurement procedures to determine the measurable SLO values. The measurements are used by the proposed framework to validate the SLOs compliance to the contracted values in the secSLA. Furthermore, we establish a monitoring scheme for validating the compliance of the service to the secSLA.

3. Enforcing the secSLA using smart contract deployed over the Ethereum blockchain. The smart contract receives and aggregates the monitoring logs corresponding to measured SLOs. The aggregation schemes are fully decentralized and are not prone to single point of failure.

4. Implementing and evaluating the functionality and performance of the monitoring scheme on Amazon Elastic Compute Cloud (Amazon EC2) instances [Ama17].

4.2 BACKGROUND

The blockchain is a novel technology that was originally invented in 2008 by Satoshi Nakamoto [Nako8]. Blockchain is a distributed public ledger storing all cryptocurrency transactions whether it's Bitcoin transactions or any other digital currency. The transactions are stored in blocks and these blocks are cryptographically chained together forming the blockchain. Unlike traditional systems which require users to trust third parties for its operation, the blockchain enables a trustless environment, where no trusted third-party exists.

4.2.1 *Consensus Algorithms*

In order for the blockchain technology to enable a trustless environment, a consensus mechanism is needed where all participating nodes in the blockchain network adhere to. The most prominent consensus mechanism widely adopted is the proof-of-work algorithm *PoW* which is introduced by Bitcoin. Ethereum uses an improved version of it, called *GHOST*, which works on the same premise. The POW is a puzzle competition which allows the first node to find a random number, called nonce, the right to propose the next block in the blockchain. Using this nonce, the hash of the entire block becomes lower than the current difficulty target.

The blockchain difficulty target is used to adjust the average time spent by miner nodes to provide the proof of work solution. The average time needed by a miner node in the bitcoin blockchain is approximately 10 mins, while for Ethereum (cf., Section 4.2.3) is approximately 15 seconds. Once a miner node succeeds in finding the right nonce, the node broadcasts the block to the blockchain network where every node checks the validity of the block.

Despite the fact that PoW algorithm is widely adopted, it still suffer from energy inefficiency. The miners are incentivized by cryptocurrency rewards, when they successfully solve the complex proof of work computations. However these computations consume huge amounts of electrical energy, that even considered more than some countries energy consumption [Ene].

4.2.2 *Types of Blockchains*

4.2.2.1 *Public Blockchains*

A public blockchain allows anyone to participate in the blockchain through sending and reading valid transactions or even in the consensus mechanism by adding new blocks to the public blockchain. Public blockchains are usually called permissionless blockchains, as every participant has the same access rights. Public blockchains are also considered to be fully decentralized [Blo] as there is no specific entity managing the participants.

4.2.2.2 *Federated Blockchains or Consortium Blockchains*

Consortium blockchains can be interesting for a group of organizations that need to share information while enjoying the data immutability feature of the blockchain. These blockchains are usually considered partially decentralized as they operate under the leadership of a group [Blo]. Consortium blockchains do not allow any person with access to the internet to participate in the process of verifying transactions.

A real example on consortium blockchains is the EU federated Cloud project, in which 11 different organizations, such as IBM,PWC, University of Southampton, cooperate together. Access control management and other key requirements in this project are handled through a consortium blockchain [SSNM16].

4.2.2.3 *Private Blockchains*

The consensus mechanism is controlled by only one entity. This means that all the write permissions on the blockchain are actually centralized to that entity [Blo]. Read permissions may be public or restricted to an arbitrary extent.

4.2.3 *Ethereum Blockchain*

Ethereum was founded by Vitalik Buterin, a cryptocurrency researcher and programmer. Unlike Bitcoin which is utilized only as digital decentralized currency, Ethereum is a distributed computing platform. Using the computing platform turing-complete programming language, Ethereum enables complex computations to be executed in the blockchain [Woo14]. Ethereum has its own cryptocurrency called ether *ETH*. One ether can be divided into smaller units of currency, where one ether corresponds to 10^3 finney, 10^6 szabo, and 10^{18} wei. Ethereum allows developers to develop smart contracts and decentralized applications "Dapps" [Eth]. The following sections present the most important aspects of the Ethereum blockchain.

4.2.3.1 *Ethereum Accounts*

There are two main types of accounts in Ethereum:

- **Externally Owned Account (EOA):** This account is held by external actors, in other words, it's controlled by private keys of the account's creator and hence transactions must be signed with the corresponding private key before being stored in the blockchain.
- **Contract Account:** Is an ccount controlled by the source code of the smart contract which defines its behaviour. When this account receives a transaction, it starts to execute the corresponding snippet of code and hence changes its state after a successful execution. This type of accounts can also interact with another contract account or an EOA.

4.2.3.2 *Ethereum Virtual Machine*

Ethereum Virtual Machine "EVM" is the runtime environment for Ethereum smart contracts. The EVM executes the smart contracts in an isolated fashion, in other words, the execution has no access to network, file-system or other processes [Evm]. This makes the execution results deterministic and any malicious node trying to alter the execution results can be easily detected. Computations in the EVM are done via a stack-based bytcode language similar to traditional assembly languages, where the programs are composed of operational codes (opcodes). Each opocode has a predefined execution cost in a unit called "Gas", where gas is the internal pricing for running a transaction or contract in Ethereum.

4.2.3.3 *Smart Contracts*

Smart contracts are basically the programs that reside in the Ethereum blockchain network and are executed by miner nodes. Any smart contract has set of functions that define its behaviour and data that defines its state [Sma]. For every smart contract created and deployed over the Ethereum blockchain, there exists a contract account. Each contract account has an address which allows users to interact with the contract by invoking its functions.

Fortunately, smart contracts are not directly written in the EVM language but actually are written in high level language such as Solidity. The code is then compiled into bytecode to be deployed over the blockchain and gets executed by the miner nodes EVM.

4.2.3.4 *How to Interact with Smart Contracts*

The first step towards interacting with a smart contract is running an Ethereum node connected to the public Ethereum blockchain. Geth is the official Ethereum node implementation that can be installed in order to connect to the public Ethereum blockchain [Get]. There are two main components that are crucial to interact with the smart contracts deployed over the blockchain, namely the Web3 API and oracles.

- **Web3 API**: The Web3 API is the most widely used API by decentralized applications, it offers the Ethereum nodes to interact smoothly with the blockchain.
- **Oracles**: By design the smart contracts are passive when it comes to the interaction with the outside world "outside the blockchain". This means smart contracts are not allowed to fetch data from external sources and need trustworthy data feeders to feed them with the required data. An oracle is basically a data feeder that collects real-world data and send it to the smart contract, hence it plays the role of an intermediary between the outside world and the blockchain. Oracles are currently provided by third-party trusted services that collect various data types such as temperature, stock market prices, etc. Oracles can utilize the web3 API in order to be able to interact with the smart contracts deployed on the blockchain [Ora].

4.3 SECSLA COMPLIANCE MONITORING FRAMEWORK

After finding the best matching CSP in Chapter 2, the SLOs associated with the secSLA are negotiated between the CSP and the customer during the secSLA negotiation phase (i.e., the negotiation phase is not covered in this dissertation). At the end of the negotiation phase, the CSP commits to deliver the agreed-on SLOs in the secSLA. Afterwards, the customer utilizes the proposed monitoring approach to validate the agreed-on SLOs compliance (shown in Figure 4.1).

4.3.1 *Monitoring Approach Architecture*

The proposed approach is composed of the following three progressive stages.

Satge (1) The agreed-on SLOs' values are extracted from the secSLA and the measurement technique for each SLO is defined. The defined measurement techniques are formulated as "tests" to be executed during the monitoring scheme. These tests are used to determine the real-time values of the SLOs. Both the contracted SLOs and tests are developed in the smart contract. The smart contract is deployed and executed in the blockchain.

Satge (2) The CSP services/SLOs are monitored over a certain period of time using customer data oracle as shown in Figure 4.1.

Satge (3) The monitoring logs measured in Stage (2) along with the measurement techniques defined in Stage (1) are used to validate the contacted SLOs; whereas the contracted values from the secSLA are used as the validation reference.

Before detailing each of the three monitoring stages, we explain the main components of our model as depicted in Figure 4.1:

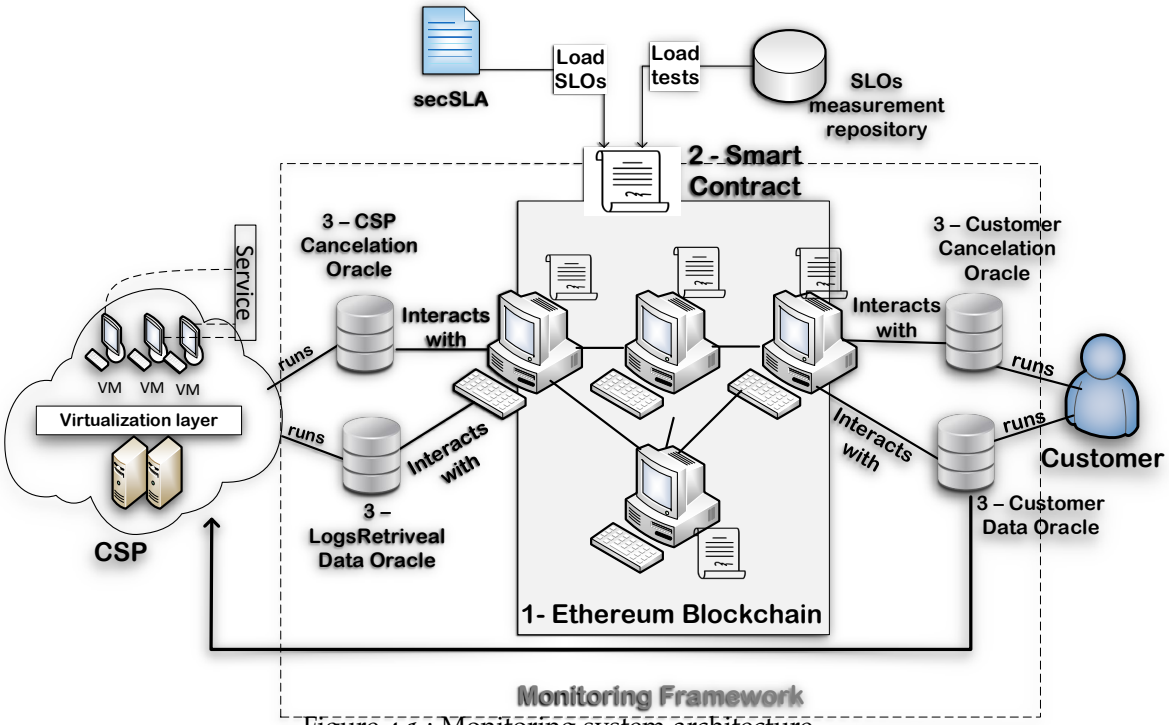

Figure 4.1.: Monitoring system architecture

1. Ethereum private blockchain is the underlying infrastructure of our system. It serves as a decentralized trustless computing platform, where our smart contract is executed in a decentralized manner. The execution results are verified and stored by every blockchain node.

2. The smart contract is the key component of our proposed approach. The smart contract includes the contacted secSLA SLOs and the measurement techniques of each SLO. Furthermore, the smart contract is responsible of:
   - Interacting with different oracles (e.g., receiving the monitoring logs from data oracles).
   - Aggregating the monitoring logs using the defined measurement techniques (where each SLO value is validated against its contracted SLO value).
   - Autonomous compensation in case of secSLA violation.
   - Offering both time-based and on demand secSLA cancellation.

3. Oracles which represent the implemented off-chain software, which depends heavily on the Ethereum web3 API and certainly can only interact with the blockchain through a valid Ethereum EOA account. In the proposed approach, there are two types of oracles (a) data oracles, and (b) cancellation oracles. The customer data oracle (shown in Figure 4.1) is responsible for:
   - Executing the monitoring tests according to a predefined monitoring frequency to report the values of the SLOs.
   - Preprocessing and batching the monitored logs to be understandable by the deployed smart contract.
   - Sending the preprocessed batched logs along with the secSLA compliance session parameters to the smart contract (e.g., number of batches, oracleID)

The log-retrieval data oracle (shown in Figure 4.1) is responsible of retrieving back the batched logs based on transaction ID from the blockchain.The cancellation oracles is used in the smart contract cancellation mechanism. Note that, oracles can only interact with the blockchain through a valid Ethereum EOA account.

We do mention that, both the CSP and the Cloud customer are running nodes in the same Ethereum blockchain and can interact with the deployed smart contract via their corresponding oracles as depicted in Figure 4.1. No other entity has access to the smart contract's functions as our developed smart contract only accepts transactions originating from either the CSP address or the customer's address. Both the CSP and the customer must agree on all the used oracles and their responsibilities beforehand. Each monitoring stage is detailed in the following sections.

4.3.2 *Stage (1): Measurement Definitions*

The SLOs contacted in the secSLA have to be monitored throughout the full service life cycle in order to assess the compliance of the CSP to the security level of these contracted SLOs. In this stage, the process of defining SLO measurement techniques is specified. The description as well as the metric of each SLO are studied to define an appropriate measurement to determine the SLO's value. The measurement definition describes the process used to determine the value of each SLO from the customer side. 142 SLOs were examined (provided by NIST [NIS08], Center of Internet and Security (CIS) [Cis], EC CUMULUS [PHK+13], EC A4Cloud [NG14] and EC SPECS [Pro14]). Out of all the examined SLOs, 9.4% can be measured from the customer side. The measurement can yield exact values which can be directly compared to the values provided by the CSP. The small number of the SLOs which can be measured is due to the limited visibility and control of the customer on the Cloud model. The majority of the proposed SLOs requires access to the Cloud platform. We used three different SLOs in our evaluation and validation experiments, namely percentage of uptime, percentage of processed requests, and secure cookies forced. We explain each of these SLOs and their measurement techniques as follows:

4.3.2.1 *Percentage of Uptime*

It is the percentage of time slots in which the service is considered available. Accordingly, the measurement period is divided into timeslots of fixed length *slotSize*. A slot is considered available if the percentage of failed requests within a timeslot is less than a defined percentage *timeslotFailThreshold*.

Measurement. To monitor the availability of the service throughout the full life cycle, requests are sent periodically to the service with a predefined frequency and the response of the CSP is verified. The status of the requests are used to calculate the percentage of uptime using Equation 4.1.

$$\frac{\sum Available\ slots}{\sum Slots} \tag{4.1}$$

4.3.2.2 *Percentage of Processed Requests*

A request is considered successful, if the service was delivered without an error and within a predefined time frame.
Measurement To measure this SLO, service requests are generated at a predefined frequency and the percentage of successful requests is calculated over the measurement period.

$$\frac{\sum \text{Successful requests}}{\sum \text{Requests}} \tag{4.2}$$

4.3.2.3 *Secure Cookies Forced*

Secure cookies forced SLO [Pro14] reports whether the service enforces the usage of secure cookies or not. This SLO is used to serve sensitive data protection, for protecting data in transmission. The secure attribute is set for a cookie to restrict sending it over secure channels only (HTTPS connections) [Bar11].
Measurement. By sending HTTP GET request to the service and examining the Set-Cookie header in the responses, one can check if the secure attribute is set to true in order to validate the usage of secure cookies.

As previously mentioned, the measurement for each SLO is formulated as tests and developed in the smart contract.

4.3.3 *Stage (2): Monitoring Approach*

In this stage we introduce the Cloud monitoring approach that enables customers to validate the compliance of a running Cloud service to its secSLA as depicted in Figure 4.2. The monitoring and validation processes comprise the following phases:

- Phase 0
  - Private Ethereum blockchain is created and both the CSP and the customer exchange their Ethereum addresses.
  - The CSP deploys the smart contract with an ether deposit value equals to the customer subscription's value and includes the customer's address.
  - The smart contract's constructor function is autonomously called on successful deployment to set its balance to the received ether and to mark the start time of the secSLA.
  - After a successful deployment, the CSP shares the smart contract's address with the corresponding Cloud customer.
  - The smart contract contains the contracted SLO values as well as each SLO measurement technique defined in Section 4.3.2.
- Phase 1
  - The customer initiates a validation session by sending a validation request to his/her data oracle. The request includes some information about the CSP (e.g., IP address or host-name) and the SLO(s) required to be validated.

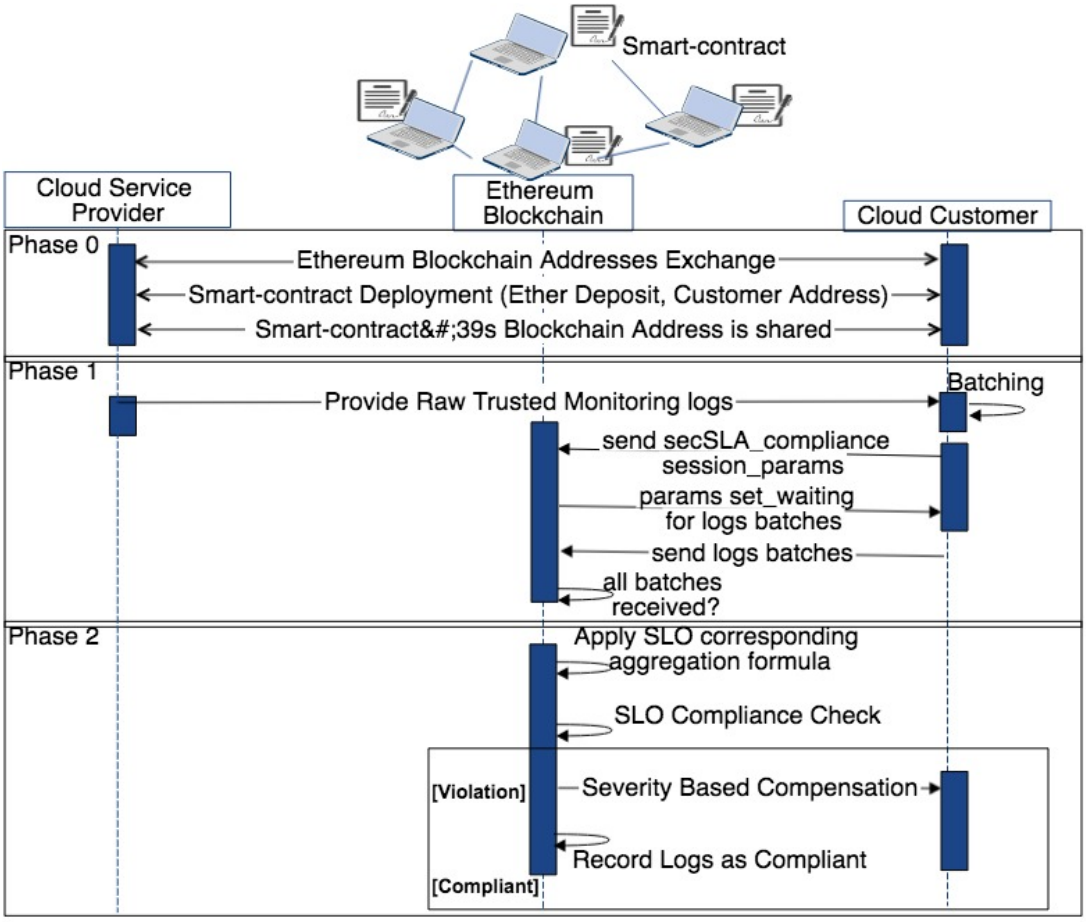

Figure 4.2.: System workflow phases - sequence diagram

- The customer's data oracle is responsible of: (a) executing the tests according to a predefined monitoring frequency to report the values of the measured SLO(s), and (b) batching the monitoring logs and then preprocessing the batched logs to match data formats that can be manipulated by the smart contract.
- The customer's data oracle interacts with the corresponding function of the smart contract (e.g., set_uptimeAvailability_sessionParams()) by sending the total number of batches and its oracle ID.
- The smart contract grants the customer's data oracle access to the corresponding receiving logs function (e.g., receive_uptimeAvailabiltyLogs()) only if there is no other oracle already interacting with the same function. This feature behaves exactly like a mutex and synchronizes the access of the distributed customer data oracles to the different smart contract's receiving logs functions.
- The smart contract receives the batches from the customer's oracle and once all the agreed on batches (e.g., 30 batches for one month monitoring logs) have been received, the contract starts the validation phase (Phase 2).

- Phase 2
  - The smart contract starts "aggregating" the received batches that constitute the whole monitoring logs session (e.g., one month). Smart contract aggregates the

received monitoring logs according to the measurement technique of each SLO in order to obtain the SLO measurement value. This value is validated against the corresponding contracted SLO value developed in the smart contract in the initial phase.

- The smart contract performs an SLO compliance by validating the aggregated values against the SLO values from the secSLA.
- In case of a detected violation, the smart contract compensates the customer based on the severity of the violation. The equivalent compensation value in ether is automatically sent to the customer address.
- If the aggregated values are in adherence to the agreed on SLOs, the smart contract records the incidence as compliant logs.
- Phase 1 and Phase 2 are repeated until, the contract's end date arrives or the smart contract receives an early contract cancellation from both parties (the CSP and the customer).

The proposed monitoring framework enables the customer to adjust the monitoring configuration (e.g., monitoring frequency, monitoring duration, validation period) according to his/her required monitoring coverage. All the processes performed by the framework are automated.

4.3.4 *Stage (3): Monitoring System Processes*

The smart contract executes two main processes which are:

4.3.4.1 *Aggregation and Compensation Processes*

1. The smart contract starts a secSLA enforcement session only when the authorized customer[1] sends to the smart contract the session parameters.
2. Before setting the session parameters, the smart contract checks first whether the corresponding aggregation function is free (in case the customer is running distributed oracles as will be explained in Section 4.3.5).
3. To verify the logs freshness, the smart contract examines the session ID, which is a unique value which represents the secSLA enforcement session and relates to the monitoring logs. If the session ID has not been recogonized (not stored in the sessionIDs contract's storage), this means the logs are fresh and eligible for an secSLA enforcement session.
4. The smart contract keeps on receiving the logs batches and once the number of batches received are equal to the contracted number of batches (e.g., thirty batches for one month logs), it starts aggregating the received batches using the corresponding measurement formula and stores the aggregated result.
5. The aggregated result is validated by the smart contract against its corresponding SLO. If the aggregated result is not compliant to its contracted SLO, the customer is compensated via Ethereum blockchain transaction originating from the smart contract (deducted from the CSP's deposit paid on deployment).

1 The customer's Ethereum address matches the Cloud customer's address stored in the smart contract

4.3.4.2 *Cancellation Process*

We propose an on-demand synchronized cancellation mechanism for the CSP and the customer, in case they both decided to cancel the secSLA enforcement smart contract before its intended termination date. Cancellation function entails two main steps: (a) receives from the CSP/customer their request to cancel the contract, and (b) sends the deposit back to the CSP (or the remainder) and then deactivate permanently the smart contract. The cancellation function only enforces the cancellation, when it receives from both the CSP and the customer a cancellation request.

Apart from the on-demand cancellation, the proposed approach also implements a time-based cancellation mechanism to adhere to the traditional agreements that define start/end date. We utilize the block timestamp, in which the smart contract was mined at, in order to mark the start date of the contract. Beside that, we rely on the date data-type in the solidity programming language to set beforehand the termination date of the smart contract.

4.3.5 *Distributed Customer's Data Oracles*

Despite the fact that, the blockchain technology provides a decentralization execution platform which is resilient against single point of failure. However, the customer data oracle proposed is actually prone to single point of failure, as it is considered to be a centralized entity. Thus, we propose a distributed customer data oracles scheme, where the customer can interact with the smart contract through several data oracles to increase the system's robustness.

To ensure the execution correctness of the aggregation mechanisms in case of distributed data oracles, a synchronization mechanism is required. The synchronization is implemented in the smart contract and it's based on the concept of mutual exclusion (mutex) where the functions responsible for receiving and aggregating the corresponding logs are considered as critical sections in which no simultaneous access of oracles is granted. This ensures correct execution of the different aggregation sessions, for instance if the first oracle has already started sending uptime-availability SLO logs to the "receive-uptime-and-aggregate" smart contract function. Other oracles are blocked to this function and can only interact with other functions (i.e., other SLOs aggregating functions). The sequence flow diagram in Figure 4.3 depicts how the distributed customer data oracles scheme works in practice. As depicted in Figure 4.3, the first oracle sends the SLO logs and thus already blocked the corresponding SLO aggregation function. Accordingly, the second oracle started to interact with the second SLO aggregate function (i.e., SLO2 function) and the third with the third SLO function.

4.4 SECURITY VULNERABILITIES OF ETHEREUM SMART CONTRACTS

Software vulnerabilities are very destructive in the context of smart contracts, as they can result in losing control of customers' digital wallets containing ether. This section discusses a duplicate compensation scenario. Duplicate compensation is the process of compensating a customer more than once based on the same monitoring session. Thus, the deployed smart contract has to keep track of the successfully aggregated monitoring sessions to prevent duplicate compensations.

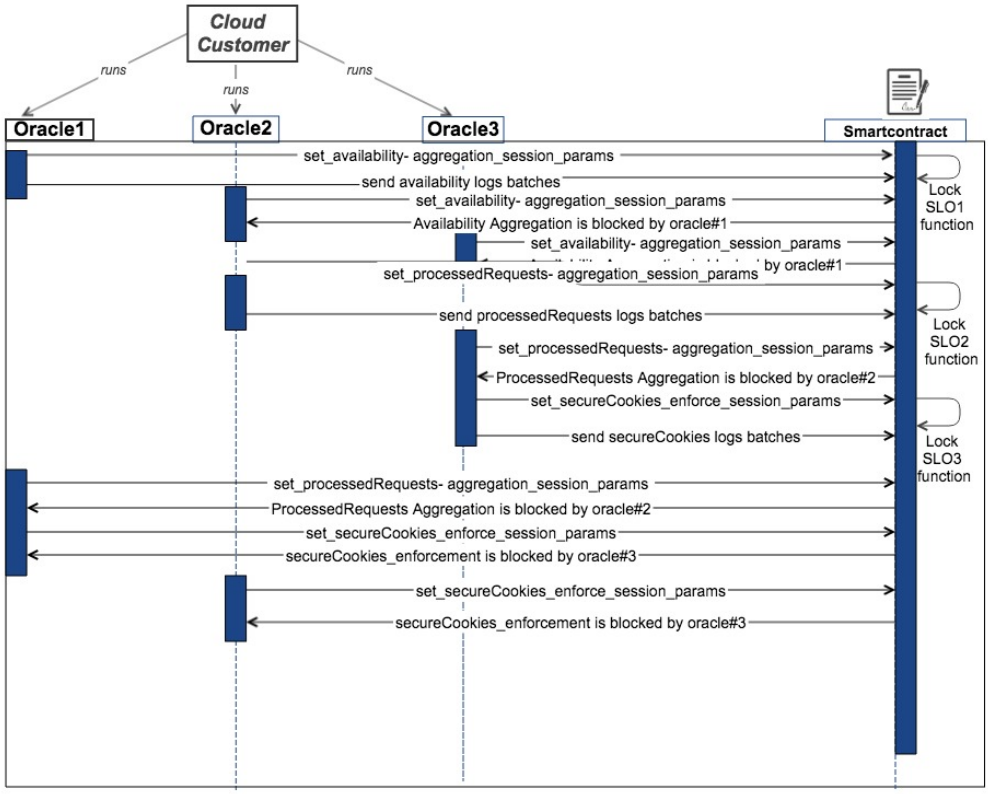

Figure 4.3.: Distributed oracles scheme workflow - single point of failure avoidance

To avoid this, the proposed system includes a unique session ID attribute which is basically used to uniquely identify the different monitoring sessions by the customer's data oracle. The session ID is sent to the smart contract along with the monitoring logs. The session ID is used by the smart contract for the aggregation, compliance tests and compensation processes.

There are three session types in the proposed approach which are related to the three used SLOs. Each session has its own session ID variable declared in the smart contract. The smart contract assigns its session ID variables to the corresponding received IDs and marks the session ID as a successfully aggregated session only when the whole batches are received and the aggregation process along with the compliance and the compensation processes are successfully executed.

4.5 IMPLEMENTATION AND EVALUATION

The proposed approach is evaluated by conducting two experiments. In the first experiment, we evaluate the functionality of the approach on Amazon EC2. The second experiment measures and compares the overall gas consumed by the oracle (with different batch sizes).

In these experiments, a machine with the following specifications is used: 4 GB RAM, Intel Core i5-3337U CPU 1.80 GHz x4, Ubuntu 16.04 LTS, OS-type 64 bit, disk 60 GB.

4.5.1 *Setting-up Ethereum blockchain*

Due to the need of high storage requirements to run a geth Ethereum node and connect to a public test-network[2] as well the intensive computational power needed for mining, we execute our evaluation and analysis using the testrpc Ethereum blockchain network simulation. TestRPC is a node.js implementation of the Ethereum protocol which offers web3 API for the interaction with the blockchain. However, it only simulates the mining process which requires high computational resources. Testrpc provides ten external Ethereum accounts each one with a hundred ether.

We implement three different aggregation mechanisms according to the SLOs specified in Section 4.3.2 (i.e., uptime-availability, percentage of processed requests, and secure cookies forced). Further, we develop a synchronization mechanism for the interleaving oracles. The on-demand and the time-based cancellation is also assured through the smart contract. Finally, we develop a compensation mechanism. The developed smart contract functions as well as the smart contract deployment process are presented in the following subsections.

4.5.1.1 *Smart Contract Functions*

The deployed smart contract is composed of the following functions:

1. SecSLA enforcement smart contract constructor, the smart contract constructor is called automatically when the contract is successfully deployed.
2. SLO related functions, which are implemented to receive the SLOs' logs from the customer data oracle, and also synchronize the access to these functions in case of interacting with multiple oracles.
3. On-demand cancellation function, which deactivates permanently the deployed smart contract.

4.5.2 *Cloud Customer Data Oracle*

The Cloud customer data oracle is implemented in python programming language. To monitor the service provider, firstly the information about the service to be monitored must be provided, including the IP address of the service and the hostname of the web based management interface. Secondly, the customer specifies the duration of the monitoring session, that is the duration over which the service is validated. In addition, the frequency at which the measurements of the SLO are conducted during the monitoring session is specified. The higher the chosen frequency the more fine grained is the measurement.

Using these specification, as well as the SLOs' values, the monitoring session is initiated. Using the frequency specified by the customer, the monitoring framework measures the value of the SLOs' by continuously monitoring the provided service. Then, the monitoring logs (of the three SLOs) are batched according to both the slot and batch sizes. These batches' feed the smart contract using a smart contract handler class. The constructor

2 Examples of public Ethereum test-networks are ropsten [Rop] (so far more than 2.7 million blocks with almost 16 million transactions) and rinkeby [Rin] (so far almost 2 million blocks with more than 4 million transactions)

function of this class is responsible for: (a) creating a web3 API object that connects to the customer's Ethereum node, (b) instantiate the deployed contract via its application binary interface (ABI JSON file) and the contract's Ethereum address, and (c) allows the customer's address to run the oracle. Using the monitoring logs and the measurement mechanisms specified in Section 4.3.2, the smart contract aggregates and validates the values of the SLOs with the built in contacted SLOs.

Finally, a batch retrieval script is developed to be used by either the CSP or the Cloud customer, in order to read back the batches stored on the blockchain by the smart contract's aggregation functions. It can be used, for instance, by the CSP to make sure that the batches sent by the customer to the blockchain are exactly the same as the logs the CSP provided.

4.5.3 *Experiment 1: Evaluating the Functionality of the Approach on Amazon EC2*

Three EC2 instances each in a different region, along with their web based management interfaces (Amazon management console) were monitored. The configurations for the instances are shown in Table 4.1.

	US_EC2	**FRA_EC2**	**TYK_EC2**
Region	US East(N. Virginia)	EU(Frankfurt)	Asia Pacific(Tokyo)
Instance type	t2.nano		
Amazon M/C Image	Amazon Linux AMI 2016.09.0 (HVM) [Ama17]		
Network	Default		
Availability zone	us-east-1d	eu-central-1b	ap-northeast-1a
Tenancy	Shared		

Table 4.1.: Instances configurations

The SLA defined by Amazon for EC2 only includes service availability. For Amazon EC2, unavailable means that all of your running instances have no external connectivity. The value committed in the SLA for this SLO is a lower bound of 99.95%.

The oracle frequently sends Internet Control Message Protocol (ICMP) echo requests to check if the service is available. With the aim to detect the shortest possible change and thereby achieve maximal coverage of changes in the monitored SLO values. For availability, the shortest possible change was assumed to be the reboot time of an instance during which the instance is unavailable. The reboot time of an instance has been experimentally assessed to be 5 seconds. Hence, the monitoring frequency is set to one test every 5 seconds. For the other security properties of the management subsystem, a lower monitoring frequency was used, since changes in the configuration of the web server hosting the console are expected to occur less frequently. Moreover, automated access to the console at high rates might be considered malicious and thus, the requests might get blocked by the CSP. Accordingly, the monitoring frequency for the consoles was configured as 6 times per hour.

The experiment was run for a month with slot size of one hour for both the uptime-availability and the percentage of processed requests service level indicators, and a slot size of 10 minutes for the secure cookies forced.

Table 4.2 shows the measured values of the SLOs for all three instances. According to the results, the instance deployed in the US East (New Virginia) offered the least *percentage of uptime* with a percentage still greater than 99%. The lowest *percentage of processed requests* is reported for the instance located in Asia Pacific (Tokyo).

We analyzed the collected data to investigate the frequencies and duration of the outages during the measurement period. The shortest observed outage is the failure of a single request, i.e., an outage duration of less than 10 seconds. The longest outage duration extracted from the results of all instances is 255 seconds experienced in the US_EC2 instance. Finally, all the consoles enabled secure cookies.

	US_EC2	**FRA_EC2**	**TKY_EC2**
% of Uptime	99.0915%	100%	99.9303%
% of Processed Requests	99.8088%	99.9930%	99.7571%

Table 4.2.: Amazon EC2 instances results

4.5.4 *Experiment 2: Consumed Gas (Cost) Evaluation*

In this experiment, we compare the overall gas consumed by our approach according to different batch sizes. As already mentioned, the SLOs' monitoring logs are batched according to the slot size as well as the batch size and then feed to the smart contract. We evaluate our approach based on sending (a) daily monitoring logs (one day batch), (b) three days batch, and (c) five days batch, to the smart contract.

A python script is developed to calculate the total consumed gas by the secSLA compliance validation and compensation for one month raw logs as depicted in Table 4.3. The total gas consumed by the approach is depicted in Table 4.3. These results are calculated based on the current estimated gas price which is(10^{-9}) ether at the ether current market value[3] (1 ether is 531$). It is worth noting that, the cost of the secure cookies enforcement session cost for the five days batch scenario cannot be measured, due to the huge transaction size which exceeds the block limit (6721975 gas). This huge size comes from the fact that the secure cookies logs were collected at a higher frequency (10 minutes slots). The number of slots comprising a one day of secure cookies logs is 6 times larger than the uptime-availability and the processed requests.

Furthermore, we calculate the amount of gas consumed by every transaction sent from the customer's oracle to the smart contract. Table 4.4 shows the average cost per transaction for each SLO's batch sizes. The web3 estimate transaction gas function is used to estimate the cost of one transaction holding 5 days of raw secure cookies logs.

3 https://ethereumprice.org/

Batch size	Total Gas Consumed	Equivalent in Ether	USD Equivalent
1 day	58848167	0.0588	31.2$
3 days	57360993	0.0573	30.4$
5 days	58283227	0.05828	approx. 29.2$

Table 4.3.: Total gas consumed by the validation and compensation processes based on one month logs with different batch sizes

Average cost per transaction			
Service Level Indicator	1 day of logs per Tx	3 days of logs per Tx	5 days of logs per Tx
Uptime-Availability	337705	951055	1661498
Percentage of processed requests	348743	989013	1725654
Secure-cookies forced	1275156	3796030	>6721975

Table 4.4.: The consumed Gas of the validation and compensation processes per SLO

4.6 RELATED WORK

Multiple approaches have been proposed to validate the compliance of the service to the SLA, by verifying the enforcement of the contracted properties. Haq et al. [HBS10] proposed a framework to manage SLA validation by defining rules to map high-level SLOs to low-level metrics that the CSP can monitor throughout the service life-cycle. Furthermore, Rak et al. [RVEE+11] and Liu et al. [LKL13] proposes Cloud application monitoring tools that detect SLA violations. Although these approaches provide effective techniques to detect SLA violations, they are only focused on (as well as most other existing monitoring techniques, such as Rana et al. [RWQ+08] and Zhang et al. [ZLLL14]) monitoring *performance* properties rather than *security* properties.

Few approaches that are concerned with monitoring security properties of Cloud services have been proposed in the literature. Ullah et al. [UA14] proposed an SLA management solution that installs monitoring agents on the Cloud service to measure service security properties. Ullah et al. [UAY13] and Majumdar et al. [MMW+15] proposed tools to be used by the CSP to audit the compliance of their services to some security properties by depending on log analysis. Nevertheless, all of the previously mentioned approaches address CSP side validation by proposing solutions managed/deployed by/at the CSP or require cooperation from the CSP. Hence, customers cannot validate the effectiveness of these approaches or guarantee transparency of the reported results.

Few approaches have been proposed to utilize the blockchain technology in Cloud computing. Margheri et al. [MFYS17] proposed a design and implementation of a governance approach for the Federated Cloud as-a-Service (FaaS) European union project (sunfish) [SSNM16]. Their proposed approach utilizes the blockchain technology as an infrastructure to ensure a distributed and democratic governance. Gaetani et al. [GAB+17] proposed an approach for tackling the database integrity challenges in the Cloud environment through utilizing the blockchain technology. Shafagh et al. [SBHD17] proposed a new paradigm for managing IoT data in the Cloud. Liu et al. [LYC+17] proposed a

blockchain based approach to replace the data integrity services provided by the Cloud service providers on the IoT data. Ferdous et al. [FMP+17] proposed a decentralized approach for runtime access control systems in Cloud federations. Lee et al. [Lee18] proposed a new approach for identity and authentication management as-a-service (IDaaS) offered by Cloud providers to their customers.

4.7 SUMMARY

In this chapter, the compliance validation framework is presented. A decentralized monitoring framework relying on Ethereum block chain to validate the compliance of the service to the measurable SLOs is proposed. The measurements for the SLOs were defined. Each measurement proposes the complete process to be carried out to obtain the value of the SLO. The defined measurements were used in the framework to test the compliance of the service to the contracted values of the SLOs in real-time. The framework enforces the secSLA using Ethereum smart contact and validates the compliance of services throughout its full life-cycle. The framework autonomously compensate customers upon services violation. The design of the framework and the role of each component in the validation process were presented. Moreover, the implementation details of the components in the framework as well as the evaluation experiments were discussed.

Part IV

USE-CASE

5

QRES: QUANTITATIVE REASONING ON ENCRYPTED SECURITY SLAS

Despite of the benefits of enclosing security-related information in the secSLAs, as discussed in the previous chapters, CSPs do not release sensitive information for security and/or commercial reasons. Publicly detailing sensitive commercial information or disclosing detailed information regarding the CSPs security posture can increase the rate of security breaches and hence, lead to financial losses stemming from reputation damages. Security posture is defined as the CSP's level of security based on the implemented security controls.

In this chapter, we present the first step towards providing security assurance and transparency between Cloud customers and CSPs, while at the same time ensuring the confidentiality of the CSPs detailed information regarding their security posture. In other words, we propose *QRES* (Quantitative Reasoning on Encrypted Security SLAs) system, that enables: (a) CSPs to disclose information regarding their security posture in their secSLAs and encrypt their secSLAs to ensure data confidentiality, and (b) customers to search for their requirements over the CSPs' encrypted secSLAs and find the best matching CSP according to their requirements.

5.1 MOTIVATION AND CONTRIBUTION

In order to design and build such a system (*QRES*), several challenges need to be addressed:

1. The Customers' cannot search over the encrypted secSLAs unless their requirements are also encrypted with the same secret key used by each CSP to encrypt its secSLA. Consequently, the system should uphold the privacy of all parties by allowing customers to search/match their requirements without neither the CSPs learning the customers' queries, nor customer obtaining the CSP's secret key.

2. The system should prevent malicious customers to ascertain information about each CSP offered services (i.e., by performing several searching iterations, each iteration with different requirements or by performing frequency analysis). Furthermore, prevent malicious CSPs from deliberately providing false information to match the customer requirements.

To tackle these problems, *QRES* utilizes a two-party privacy preserving query over an encrypted data scheme (named *QeSe*). *QeSe* is based on a privacy preserving computation (i.e., Yao garbled circuit [Yao86; LP09]) to allow CSPs to encrypt the customers' search queries without learning them. Accordingly, customers can find the best matching CSP's secSLA without knowing the CSP's encryption key. Furthermore, the protocol supports a new security property by enhancing the security setting of the traditional two-party privacy preserving computation (Yao garbled circuit evaluation). This is achieved by validating the participants' inputs before computation.

Contributions. In this chapter the following contributions are proposed:

1. *QRES*, the first system which enables CSPs to publicly disclose detailed information about their offered services in encrypted secSLAs and customers to assess the CSPs' offered security services and find those satisfying their requirements. *QRES* is implemented and benchmarked using Amazon AWS DynamoDB [Ama12]. We show that the system's performance is practical for the presented use-case. We utilize real-world CSPs' secSLAs found on the public STAR repository.
2. A two-party privacy preserving query over encrypted data scheme (named *QeSe*), used as a basis for *QRES*. The protocol enhances the security setting of the traditional two-party privacy preserving computation (Yao garbled circuit evaluation).
3. Formal proof of the system's correctness by firstly defining the needed security properties. Then conducting a formal security analysis of the proposed system using ProVerif [Bla01], an automated cryptographic protocol verifier, establishing the defined properties against a strong adversarial model (i.e., Dolev-Yao adversary model [DY83]).

5.2 BACKGROUND

This section briefly explains the basic concepts and cryptographic notations used in this chapter.

5.2.1 *Format of secSLAs*

The secSLA shown in Figure 2.1 can be specified using a machine readable format such as an XML structure as depicted in Listing 5.1.

Listing 5.1: Excerpt of the SLA depicted in Figure 2.1

```
< SLA slaid="sla1">
<service id="S1" name ="Identity and Access Management" category="IAM" pre="1">
<control id="S1.1" name ="Information Security Incident Management" category="IAM-09"
     pre="2">
<slo id="S1.1.1" name ="Percentage of timely incident reports" value="level3" pre="3"
    ></slo>
<slo id="S1.1.2" name ="Recovery time" value="level2" pre="4"'></slo>
</control></service></ SLA>
```

The XML data values can be captured using the pre-fields shown in Listing 5.1 (named "pre"), which are sequence numbers that count the open tags in an XML. An example of how the pre-fields are computed is depicted in Table 5.1. Each pre-field is used as a service identifier as each service/SLO has a unique pre-field in the secSLA.

5.2.2 *Searching Over Encrypted Data*

We define the problem of searching over encrypted data using the following example. Assume a customer who encrypts his/her documents and stores those at the CSP's storage server. However, by encrypting these documents, the customer can not search for certain keywords anymore and thus, content retrieving is very inefficient. The basic approach for retrieving the required data related to a certain keyword would require the customer to

XML structure	pre-field
$< S_1 >$	1
$< S_{1.1} >$	2
$< S_{1.1.1}$ value=$level_3 >$	3
$< S_{1.1.2}$ value=$level_2 >$	4
$< /S_{1.1} >$	

Table 5.1.: Excerpt of a secSLA XML structure with the calculation of pre-fields

download *all* stored encrypted documents, and then decrypt them to perform the keyword search. However, this solution is time consuming and impractical. In addition, retrieving all files incurs unnecessary network traffic, which is undesirable in the pay-as-you-use Cloud paradigm used today.

From the above problems, the need arises for an efficient data retrieval scheme which enables the customer to search directly over encrypted data. A solution to these problems is what is widely known as searchable encryption (SE). SE allows a customer to encrypt data in such a way that he/she can later generate search tokens to send queries to the CSP. Given these tokens, the CSP can search over the encrypted data and retrieve the required encrypted files. As specified in [KPR12], a SE scheme is secure if: (i) the ciphertext alone reveal no information about the encrypted data, (ii) the encrypted data together with a search token (i.e., queries) reveals at most the result of the search, and (iii) search tokens can only be generated using the same encryption key used to encrypt the data.

There exists a large number of SE schemes [SWP00; BCOP04; BBO07; KPR12] which are either deterministic or randomized. Deterministic schemes [BBO07] encrypt the same message to the same ciphertext. However, it does not protect against frequency analysis attacks. On the other hand, randomized schemes [SWP00; KPR12] prevent frequency analysis by salting ciphertexts and thus providing stronger security guarantees. However, the usage of salt in these schemes requires combining each token with each salt, resulting in a processing time linear in the number of salts for each token. In [SLPR15] Sherry et al., introduced an encryption scheme which achieves both the detection speed of deterministic encryption and the security of randomized encryption.

5.2.3 *Privacy Preserving Computations*

Secure two party computation. The aim of secure two-party computation is to enable both parties to carry out computing tasks without revealing information of any kind about private data to the participants. Assume two parties, A and B have some private information. They want to learn the result of some function using both of their inputs, while each party would learn nothing about the other's input. To achieve this, Yao's protocol based on garbled circuit (named Yao garbled circuit) [Yao86; LP09] is used. Yao's protocol allows two parties to exchange a garbled circuit and garbled inputs for a function, which can be used to compute an output without leaking information about their inputs.

A garbled circuit is a circuit that consists of garbled gates and their decryption tables. In a garbled gate, two random bits have been selected for every input wire to the gate, representing 0 and 1. Those bits garble the gate, making it impossible to compute the output

unless someone has access to the garbled computation table. The garbled computation table maps essentially the random inputs to the output of the gate, which is also random.

We illustrate the basic idea of how Yao's garbled circuit can be used, assume two parties A and B that have secret inputs, x and y respectively. Both of them want to compute $F(x, y)$ without revealing their private inputs to each other (x to B and y to A).

To achieve this, one party (for instance A) prepares a garbled version of the computing function F (named GarF). Basically, GarF produces the same output as $F(x, y)$, if given the corresponding encoding of each bit of both inputs x and y.

Garbling the inputs is performed by producing a pair of labels for each input bit of F (G^0, G^1, that is one label corresponds to bit 0 and the other to 1). Next, A sends GarF along with the encoding of x to B, which only needs the encoding of y from A to compute the GarF without learning any intermediate values. For this task, both parties use oblivious transfer [Rab05; NP99; ALSZ13].

Oblivious transfer (OT). OT is a crucial component of the garbled circuit approach, as it enables party B to obtain the encoding of the b bit from A, without (i) A knowing b and (ii) B learning the encoding scheme. In this way, party B can request from A the keys that he can use for his input encoding without i) A learning B' input and ii) B exploiting the protocol by having access to the encoding scheme and computing much more than allowed.

5.3 REQUIREMENTS ANALYSIS

In this section, the system model is described, the system requirements are presented, and the threat and trust models are defined.

5.3.1 *System Overview*

Finding the best matching CSP (according to the customer's security requirements) is the objective of the proposed system. Our system model (depicted in Figure 5.1) involves m CSPs[1], a customer CSC, and a broker B. The customer CSC is a company or an individual who is searching for the best provider that satisfies his/her requirements. The CSPs are Cloud providers that disclose information about the offered security posture in their secSLAs. CSPs are encrypting their secSLAs before sending them to the broker. The broker B[2] is an entity that performs the searching of the customer requirements over the CSPs' encrypted secSLAs on behalf of the customer. Hence, B ranks and manages the selection of the best matching CSP.

In our system (Figure 5.1), CSPs' encrypted secSLAs are certified and digitally signed by a trusted certification authority (i.e., auditor). The trust assumption relies on the fact that the CSPs' certificates are valid and trusted and thus, their encrypted secSLAs are verified and digitally signed by the auditor. After a successful authorization, every provider possesses a digital signature on their encrypted secSLA. The CSPs send their signed, encrypted secSLAs to an intermediate broker afterwards. After the broker verifies the auditor's signature, it stores each encrypted secSLA in a database.

Furthermore, customers send their requirements to the broker which tries to match the customer's requirements against the stored encrypted data in order to find the best match-

1 Throughout this chapter, we explain our model and queries searching scheme using only one CSP. Nevertheless, the same model applies for all CSPs.

2 We assume the case of a novice or basic customer who can not search for his/her over encrypted data

ing CSP's secSLA. During this phase, neither the broker can learn the CSP's encryption key nor the CSP's can learn the customer's requirements. Finally, the broker sends the customer the CSPs ranking according to the customer requirements.

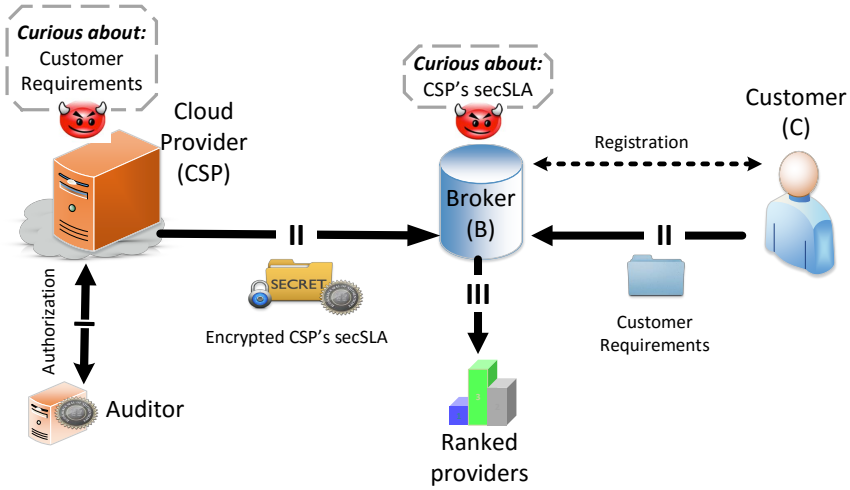

Figure 5.1.: *QRES* System Model

5.3.2 *Threat Model*

Security literature distinguishes between two adversarial model for secure computation; participants can be either semi-honest or malicious. We consider a semi-honest (also known as honest-but-curious) threat model in our system. This is a commonly used security model for secure computation (we refer the reader to Goldreich [Golo9] for details) where the semi-honest participants correctly follow the introduced protocol but attempt to obtain additional information about the other participants. By considering the semi-honest model, a dishonest participant observing the system's network should not be able to alter or recover stored data.

The semi-honest setting is relevant, as all entities, the CSP, B, and CSC, would like to continue the protocol; acquire the best matching CSP according to CSC's requirements. However, each entity can attempt to obtain additional information about the other entity's input as depicted in Table 5.2.

5.3.3 *Trust Model*

Customers can only trust the result of an assessment if the information taken as an input is reliable. In other words, in order to guarantee the validity of the proposed system, the encrypted secSLAs provided by the participating CSPs are required to be certified from a trusted certification authority (the auditor). For example, an auditor certifying the CSP's security posture as reflected by its secSLA (e.g., based on an ISO 27001 certificate [Clo17c]). The trust assumption relies on the fact that the CSPs' certificates are valid and trusted and thus, their encrypted secSLAs are verified and digitally signed. Such signing scheme

Customer CSC	Tries to learn the CSP's private key k in order to learn the CSP's secSLA
Cloud Provider CSP	Tries to learn CSC's requirements. or sends a faulty secSLA; faulty secSLAs are secSLAs which contain service levels the CSP does not offer, and could not fulfil, in order to match the customer requirements.
Broker B	Tries to (a) alter CSC's requirements in order to match a colluded CSP, or (b) collude with CSC to learn the CSPs' secSLAs.

Table 5.2.: The semi-honest threat model of our system

should provide a proving statement without revealing the secSLA input. For simplicity, we assume that the participating CSP's encrypted secSLAs are to be verified and digitally signed. We summarize the initial knowledge of each entity in Table 5.3.

Parameter	Each entity's knowledge			
	CSC	CSP	B	Auditor
CSP's secSLA		×		
CSP's Encryption Key		×		
CSP's Certificate/ID		×		×
CSC's Requirements	×		×	

Table 5.3.: Every participating entity's initial knowledge

5.3.4 *System Requirements*

In order to provide the privacy and correctness guarantees, the presented system must ensure:

1. *Input Validation:* The CSPs' encrypted secSLAs provided by the participating CSPs are digitally signed by an auditor.

2. *Data Integrity:* The system must mitigate any attempt to alter the customer's requirements. This attempt can be performed by a CSP to modify the requirements according to its secSLA.

3. *Data Confidentiality:* The encrypted CSP's secSLA can only be decrypted using its CSP's private key k. Further, the broker B should *only* learn the output of the searching queries (i.e., the search queries represent the customer requirements). Moreover, the CSP should not have access to CSC's requirements to avoid changing its secSLA specifications according to those.

   Note that, B or CSC can ensure the secSLA compliance by monitoring and verifying the offered service levels by (i) using appropriate log samples provided by the CSP, and/or (ii) adding alerts and triggers based on the service key measurable param-

eters [HD12] as explained in Chapter 4. In this chapter we only focus on the CSPs evaluation and services assessment.

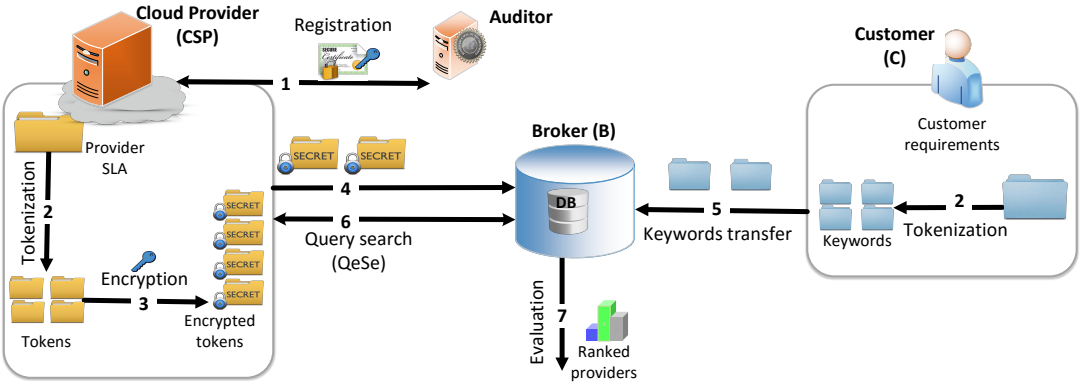

Figure 5.2.: *QRES* System Architecture

5.4 QRES ARCHITECTURE

In this section, we detail each of *QRES* progressive stages as depicted in Figure 5.2. As an initial phase the customers register and are verified by the broker B before *paying* the broker for the offered assessment service. Furthermore, we assume that the broker's certificate ($cert_B$) is validated by the auditor and is sent by the auditor to the CSPs.

Stage 1: Providers Authorization.

Every CSP is registered and verified by a trusted auditor. The auditor verifies the CSPs certificate and then signs the encrypted secSLA. The broker B can verify the validity of the encrypted secSLAs by verifying the auditor's signature.

Stage 2: Tokenization.

After CSC creates his/her set of requirements using the same secSLA-XML structure used by the CSPs to define their provided services, both the CSP and CSC tokenize their secSLA-XML data specifications such that:

- Each CSP splits its specified services (in the secSLA) into substrings. For every substring, it creates a fixed length token (8 bytes per token). The generated tokens are denoted as "t_{CSP}", such that $T_{CSP} = t^1_{CSP}, \ldots, t^n_{CSP}$; where T_{CSP} specifies the set of services offered by the CSP in its secSLA and n is the number of tokens.
- Similarly, CSC splits his/her requirements into substrings and then generates a fixed 8 bytes token for every substring. The customer's generated tokens are named "keywords" and denoted as "w_{CSC}" so that $W_{CSC} = w^1_{CSC}, \ldots, w^n_{CSC}$; where W_{CSC} is the set of customer requirements.

Both the customer and the CSP use the same secSLA template with the same services/SLOs where each of them specify different SLO values. Tokens are generated for each SLO, by searching for the SLO id and extracting only its value and pre-field of the specified SLO (i.e., each SLO in a secSLA XML has a unique pre-field as specified earlier and depicted in Table 5.1). For example, the tokens generated from Listing 5.1 are: "level3||3" and "level2||4". Therefore, for each secSLA no similar tokens can be generated as no equal pre-fields exist. Even if the CSP is offering two values for a specific SLO, both tokens would be different because of the different values.

Stage 3: Tokens Encryption.

QRES utilizes a deterministic encryption scheme $\mathsf{Enc}(k, x)$, such that the encryption of the CSP token (t_{CSP}) is denoted by $\mathsf{Enc}(k, t_{CSP})$. For instance, in order to check if the CSP's token (t_{CSP}) is matching the customer keyword (w_{CSC}), we can simply check if $\mathsf{Enc}(k, t_{CSP})$ is equal to $\mathsf{Enc}(k, w_{CSC})$. Unfortunately, deterministic encryption schemes, which are rather fast, cannot be used in every case, as every occurrence of t_{CSP} will result to the same ciphertext. However, encryption randomization is not required in our system as every generated token t_{CSP} is unique and occurs only once in the CSP's secSLA (as every token contains different pre-field). This makes the occurrence of the same token (t_{CSP}) more than once not possible.

We utilize $AES-CBC$ encryption scheme to encrypt each CSP's generated token $AES_k(t_{CSP})$. We use a typical instantiation of a hash function, which is $\mathsf{SHA256}$, to compute IV. We detail and analyze the *QRES* system implementation in Section 5.5.

Stage 4: Tokens Transfer.

Both the CSP and the customer, send their tokens to the broker. B verifies the auditor's signature and then saves the CSP's encrypted tokens in a database. For m CSPs, B saves m different lists with encrypted tokens.

Stage 5: Keywords Transfer.

In this stage, B receives CSC's requirements (i.e., keywords) securely via an encrypted channel (e.g., SSL). Once B receives CSC's messages, it saves the messages and starts a secure two-party computation with each CSP in order to find the best matching CSP according to CSC's security requirements in the next stage.

Stage 6: Query Search (QeSe).

QeSe is based on two-party privacy preserving computation (i.e., Yao garbled circuit [Yao86; LP09]) to allow CSPs to encrypt the customers' search queries without learning them. For simplicity, we explain this stage using one CSP and B. The broker B has already stored each CSP encrypted tokens ($\mathsf{Enc}(k, t_{CSP})$) (from the previous

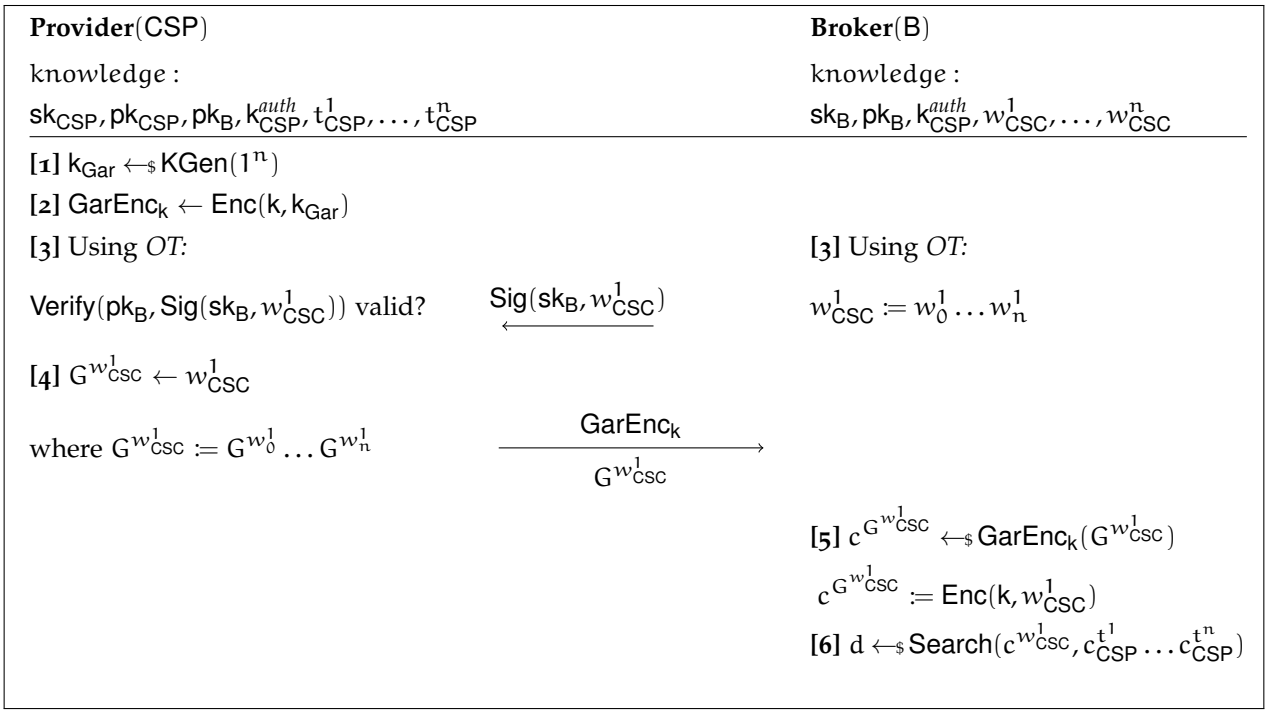

Figure 5.3.: Secure computation protocol between a CSP and the broker (QeSe).

stage) in a database. Furthermore, B has already received the customer keywords. In order to find the encrypted tokens matching the customer keywords, the customer keywords have to be encrypted by the same encryption key used by CSP to encrypt its tokens. In other words, B has to compute $Enc(k, w_{CSC})$ for every w_{CSC} using the same encryption key used by the CSP. The main challenge here is for B to obtain the encrypted w_{CSC} without knowing the CSP's secret key k and without allowing the CSP to learn w_{CSC}.

QeSe allows all parties to jointly achieve their purpose by running a secure two-party computation between the CSP and the broker. We enhance the security setting of the traditional two-party privacy preserving computation by allowing the garbled circuit to check the validity of both the encrypted tokens and the broker's digital signature of the customer's requirements. In case there is a problem with the validation our protocol quits.

The CSP provides B with a "garble" of the encryption function with the key k hard-coded in it, as depicted in Figure 5.3 (line 2) and Figure 5.4 (we denote this garbled function by $GarEnc_k$). This garbling hides the CSP's secret key k. $GarEnc_k(x)$ produces the same output of $Enc(k, x)$ if given the corresponding encoding of each bit of input x (i.e., G^x). Thus, B can run this garbled function on each keyword w_{CSC} in order to obtain the $Enc(k, w_{CSC})$ only if it is able to acquire from the CSP an encoding for the w_{CSC} ($G^{w_{CSC}}$) as depicted in line 4 in Figure 5.3. For this task, both parties (CSP and B) use an oblivious transfer protocol, where B receives an encoding of w_{CSC} from the CSP without revealing w_{CSC}. The encoding of keywords $w^1_{CSC}, \ldots, w^n_{CSC}$ is denoted as $G_1^{w^1_{CSC}}, \ldots, G_n^{w^n_{CSC}}$. Note that, on input $Sig(sk_B, w_{CSC})$, $GarEnc_k$ first checks if $Sig(sk_B, w_{CSC})$ is a valid signature of w_{CSC} using B's public

key (pk_B). If it is valid, w_{CSC} is garbled as shown in line 4 in 5.3. It is important for the security of the protocol to mention that, a garbled circuit is no longer secure if B receives more than one encoding for the same circuit. Thus, B obtains a fresh, re-encrypted garbled circuit GarEnc_k for every w_{CSC}. At the end of this process, B gets the $\mathsf{Enc}(k, w_{CSC})$ for every w_{CSC}. Afterwards, B searches for an exact matching of $\mathsf{Enc}(k, w_{CSC})$ over all CSPs encrypted tokens (as depicted in line 6 in Figure 5.3). Each successful matching result is saved in a list at B using an index d. The process is repeated for all CSPs. Hence, for m CSPs, B creates m lists. Each of these lists contains the number of encrypted tokens matching CSC's keywords.

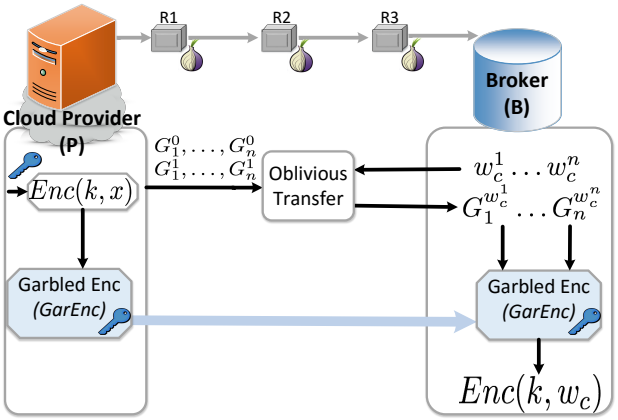

Figure 5.4.: Illustration of the secure computation protocol between the CSP and the broker (*QeSe*).

Stage 7: CSPs Ranking.

As a last step, the CSPs are evaluated and ranked according to the number of tokens matching the customer requirements. The CSP with the highest number of encrypted tokens matching the customer keywords, is given the highest score and selected as the best matching provider.

5.5 IMPLEMENTATION AND EVALUATION

Implementation. *QRES* is implemented in Java, using Apache-Tomcat 9.0 and Amazon DynamoDB [Ama12] database, which is a NoSQL database service. The DynamoDB supports both string and key value store models which we used to store the encrypted tokens (B side). We used Tomcat web server on our Ubuntu machine to transmit the encrypted data to Amazon DynamoDB (the CSP side). We implement the hash functions using HMAC based on SHA-256. The symmetric encryption scheme is implemented based on AES 128. Moreover, we use $\beta = 128$ for nonces and 8 bytes per token. Furthermore, we modified Yao's garbled circuit coding [Bre11] to fit our encryption scheme. Finally, we implemented two Java Server Page (JSP) files to send the tokens and to search for the keywords. The complete source code along with a detailed explanation of setting up and using the *QRES* can be found at [QRE17].

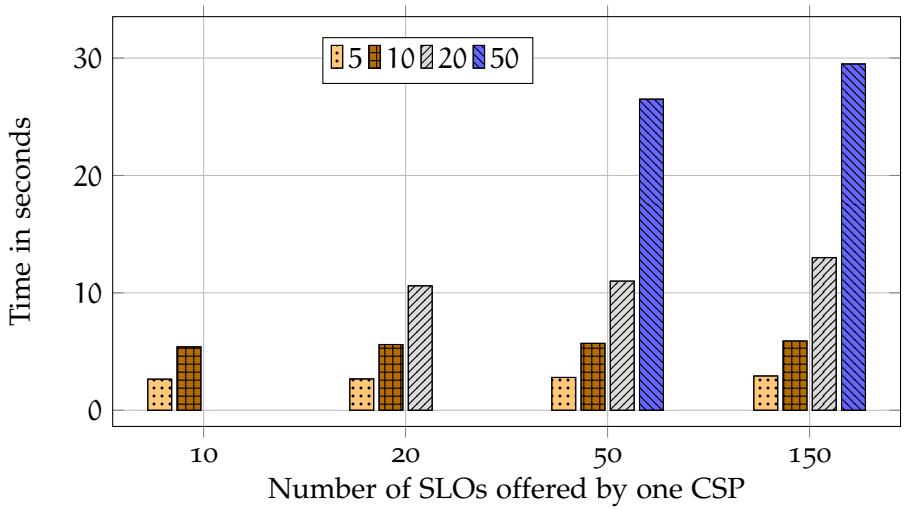

Figure 5.5.: Time used by a customer to search for her/his different keywords over varied number of SLOs offered by one CSP.

Performance Evaluation. We evaluate the performance of the *QeSe* protocol running between the CSP and the B using two use-cases. First, we use *QRES* with one CSP which is offering various tokens (SLOs) in its secSLA . Note that, the CSP is offering *one service level for each SLO* in its secSLA. Each token (8 bytes) is encrypted and sent to Amazon DynamoDB. Figure 5.5 shows the amount of time spent by a broker to search for the customer keywords (i.e. 5,10, 20 and 50 keywords) over 10, 20, 50 and 150 SLOs provided by one CSP. The use-case shows that the time to search for the customer's different keywords over various number of SLOs is almost the same despite the number of the offered SLOs. This is expected as the most time consuming part of the computation of the circuits is the same despite the amount of offered SLOs.

Further, we explore the case in which a customer compares various CSPs based on their advertised secSLAs. Figure 5.6 shows the amount of time a broker spends to search for the customer requirements (keywords) over 1, 10 and 30 CSPs' secSLAs where each secSLA consists of 150 SLOs. For each CSP's secSLA, we extracted 150 tokens (8 bytes each) which are then encrypted and sent to Amazon DynamoDB (in our current implementation we saved each CSP token in a different table). The use-case shows a linear time progression as the broker searches for more keywords in each CSP's table.

The SLOs are extracted from the public STAR repository [Clo17d]. The rationale for using STAR repository is that (i) to the best of our knowledge no other Cloud secSLA repositories are publicly available and (ii) major CSPs are still in the process of restructuring their SLAs by leveraging the recently published ISO/IEC 19086. Currently, the STAR contains reports with CSP's answers to Consensus Assessments Initiative Questionnaire [Clo17b] with yes/no answers. Furthermore, we utilized other requirements defined in multiple research projects such as A4cloud [NG14], CUMULUS [PHK+13], and SPECS [Pro14].

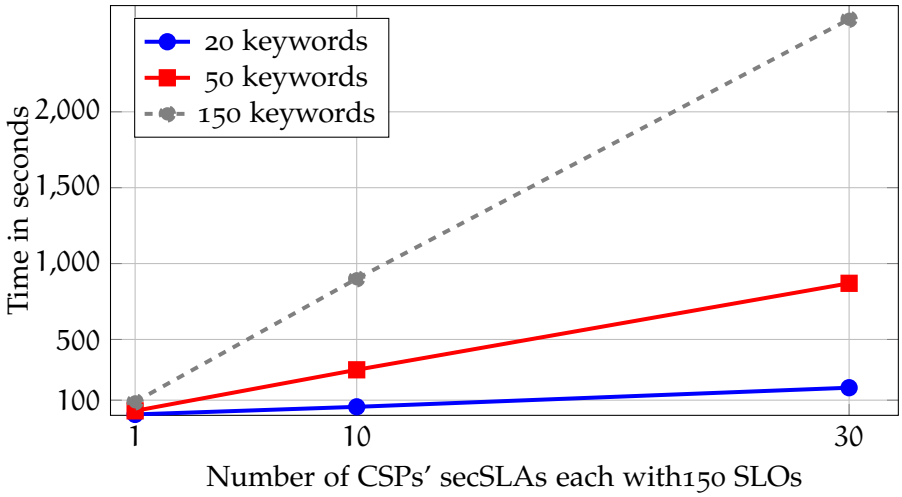

Figure 5.6.: Time used by the customers to search for their different keywords over 150 SLOs offered by varied CSPs.

5.6 SECURITY ANALYSIS

In this section we formally analyse and verify the security considerations of the system architecture described in Section 5.4, with respect to the security requirements defined in Section 5.3.4.

5.6.1 *Formal Analysis*

Formal methods can be used to model a cryptographic protocol and its security properties against an adversarial model, together with an efficient procedure to determine whether a model satisfies those properties.

In this section, we analyse the security of the proposed system protocols with respect to the security objectives outlined in Section 5.3.4 using ProVerif [Bla01]. ProVerif is an automated verification tool that can handle an unbounded number of protocol sessions. It is used to model cryptographic protocols and their security properties against a strong adversarial model. In contrast to other state-of-the-art tools (e.g., the Avispa tools [ABB+05] and Scyther [Cre08]), ProVerif provides a larger feature set [CLN09]. Furthermore, it allows the modelling of cryptographic primitives using equational theory which is used to model the Yao's garbled circuit protocol.

In order to verify a system's security, first we define a list of security properties. Then, the context in which the system functions are created. The system's context consists of assumptions about the environment and the adversarial model. We model the formal specification of the *QeSe* protocol, the security objectives and the adversary model using applied pi-calculus [Bla01]. The applied pi-calculus modelling is used as an input to the ProVerif tool. ProVerif then proves if the claimed security properties are fulfilled. The formal specification includes the following components:

1. Agent model. Agents represent the protocol parties which execute the roles of the protocol. For instance, we have the sender and the receiver roles in the proto-

col. Therefore, in each protocol, each agent performs one or more roles. The agent model is based on a "closed world assumption", which means that honest agents show no behaviour other than the one described in the protocol specification (this model corresponds to honest-but-curious threat model in Section 5.3.2).

2. Communication channel. The communication model describes how the messages are exchanged between the agents. The channel can be private or public according to the threat model. In our scenario the channel is public.

3. Threat model. Adversaries are modelled as agents that aim to violate the security objectives. ProVerif uses the standard Dolev-Yao adversary model [DY83]. In this model the adversary has complete control over the public communication network.

4. Protocol specification. The protocol specification describes the behaviour of each of the roles in the protocol. The system does not execute the actual protocol but it executes the protocol roles performed by the agents.

To model *QeSe* protocol, we model Yao's garbled circuit protocol to verify *Data confidentiality and Data Integrity*. Two agents are used to model our protocol in the applied pi-calculus (i.e., one CSP and one broker B). Based on Yao's circuit two algorithms are defined (namely garble Gar and evaluate Eval). Gar takes an input function with n bits and outputs a garbled function and a pair of labels for each input bit of the function. Eval takes the garbled function and garbled inputs and returns the required function output.

We model the CSP's encryption function as a public parameter represented by a free variables z_f (i.e., free variable is similar to global scope in programming languages; that is, free names are globally known). The private parameters of the CSP and B are the protocol inputs (the encryption key and the keyword, respectively). The CSP's and B's private parameters are represented as free variables x_{CSP} and y_{B}, respectively.

The goal of B is to obtain the encryption of its keyword using the CSP's key. To achieve this we use the protocol specification shown in Figures 5.3. The garbling function is modelled using a generated random key, named as the garbling key $\mathsf{Gar}(., k_{\mathsf{Gar}})$. The garbling key is used to securely garble the circuit at each protocol run (as stated earlier, to ensure the garbled circuit security, B receives a fresh, re-encrypted garbled circuit for each input $\mathsf{Gar}(y_{\mathsf{B}}, k_{\mathsf{Gar}})$.

To model the oblivious transfer, the Broker B generates a commitment of each keyword with a fresh generated nonce. The commitment of each keyword cannot be modified and is hidden using the nonce. The commitment is used to request a garbling of the keyword from the CSP without disclosing the keyword to the CSP. We do model the encryption output as $\mathsf{Eval}(z_f, x_{\mathsf{CSP}}, y_{\mathsf{B}})$, where both CSP and B want to find the keyword encryption output without disclosing x_{CSP} to B and y_{B} to CSP.

5. Security properties. These properties specify the security requirements of the protocol defined in Section 5.3.4.

a) ***Data Confidentiality:*** Modelling strong secrecy to verify the encrypted SLA's confidentiality is straightforward and modelled easily in ProVerif (adversary

learns nothing about the SLA). However, to prove the confidentiality property in *QeSe*, we have to prove that the only leakage about the input of the two parties (CSP and B) should come from the result of the evaluated function. In other words, if a party obtained the output of the evaluated function, no leakage should occur about the two parties' inputs.

This is proved using the indistinguishability property (the notion of indistinguishability is generally named observational equivalence in the formal model [Bla01]). Intuitively, two processes P_1 and P_2 are observationally equivalent (i.e., written $P_1 \approx P_2$), when an attacker cannot distinguish between the two. Formal definitions of the indistinguishability property can be found in [AF01; BAF08]. For example, the privacy property of an electronic voting protocol is expressed as [Bla01]:

$$P(\mathsf{sk}_A, v_1)|P(\mathsf{sk}_B, v_2) \approx P(\mathsf{sk}_A, v_2)|P(\mathsf{sk}_B, v_1)$$

P is the voting process, and an attacker cannot distinguish between the two situations; i) in which A votes for v_1 and B votes for v_2 from ii) that A votes for v_2 and B for v_1, where v_1 and v_2 are the candidates for whom A and B vote.

To prove the data confidentiality, we model two processes that are replicated an unbounded number of time and executed in parallel. In the first process P_1, CSP and B sends two inputs, represented as free variables x^0_{CSP} and y^0_B, respectively. While in the second process P_2, the CSP and B sends free variables x^1_{CSP} and y^1_B, respectively. If the two defined processes are observationally equivalent ($P_1 \approx P_2$), then we say that the attacker cannot distinguish between the two process input.

Theorem 1: Data confidentiality is preserved in the system if P_1 and P_2 are observationally equivalent ($P_1 \approx P_2$). In our protocol this is proven using ProVerif.

b) ***Result Integrity:*** In *QeSe*, data output integrity is preserved if the final result obtained by both parties CSP (sender) and B (receiver) after a secured computation is consistent. This means that for the same inputs and the same function, both CSP and B find the same result. This can be proved using the ProVerif's correspondence property [Bla02]. An example of the correspondence property is, $e(M_1, \ldots, M_j) \Longrightarrow e'(N_1, \ldots, N_k)$, where for any trace of the protocol for each occurrence of event $e(M_1, \ldots, M_j)$, there is a previous occurrence of $e'(N_1, \ldots, N_k)$.

Therefore, we define the the integrity property as follows:

$$\begin{aligned}
&\mathsf{CSP}term(z_f, x_{\mathsf{CSP}}, y_B, r) \Longrightarrow \\
&y_B = \mathsf{com}(y_1, n) \wedge r = \mathsf{Eval}(z_f, x_{\mathsf{CSP}}, y_1) \text{ and} \\
&\mathsf{B}term(z_f, x_{\mathsf{CSP}}, y_B, r') \Longrightarrow \\
&x_{\mathsf{CSP}} = \mathsf{Gar}(x_1, \mathsf{k}_{\mathsf{Gar}}) \wedge r' = \mathsf{Eval}(z_f, x_1, y_B)
\end{aligned} \tag{5.1}$$

Note that, the CSP's and B's private parameters are represented using free variables x_{CSP} and y_B as stated earlier. The property in equation 5.1 states that the CSP with arguments $z_f, x_{\mathsf{CSP}}, y_B, r$ terminates a protocol run with B after (i) CSP

receives a commitment of B's input with a random value n ($y_B = \mathsf{com}(y_1, n)$) and (ii) finding the result r of the of the function z_f on inputs x_{CSP} and y_1. Similarly, B terminates the protocol run with the CSP after receiving the CSP's garbled input x_{CSP} and finding the result r' of the garbling function z_f on inputs x_1, y_B. Using the correspondence property defined in equation 5.1, the process $!(\mathsf{CSP}(z_f, x_{CSP}))|!(\mathsf{B}(z_f, y_B))$, which models the two agents processes and can be executed any number of sessions.

Theorem 2: Data integirity of all possible executions of sessions of honest parties in process P is preserved if the correspondence property defined in equation 5.1 is true. For our protocol this is proved using Proverif.

5.6.2 *Further Security Considerations*

Our system and its protocol are designed and verified against honest but curious entities. However, with minor protocol modifications our system mitigate against malicious parties.

- **Malicious CSP:** In the case of a malicious CSP, it might send (i) an incorrect garbling function to broker B, and/or (ii) a faulty encrypted secSLA. To prevent the first form of attack (i), B can prove that the garbling is correct by using the "cut-and-choose" technique [LP07; HKE13]. Under this technique, a CSP constructs n versions of the circuit, each structured identically but garbled differently so that the keys for each gate in each circuit is unique. Additionally, the CSP generates a commitment for each of its garbled inputs. The CSP then sends each of the garbled circuits with its garbled inputs to B. Further, B selects $n-1$ versions of the circuit to verify. The CSP de-garbles each of the $n-1$ selected circuits, so that B can verify that each of the revealed circuits are constructed correctly and as expected.

  B checks whether the CSP's garbled inputs match their corresponding (previously sent commitments). If everything is correct, B evaluates the rest of the circuits and derives the output from them. Thus, if a malicious CSP constructs the circuits incorrectly, B will detect this with high probability. Regarding faulty SLA's, as explained earlier, an auditor (whose reputation depends on its trustworthiness) first ensures the validity of the CSP's input and thus prevents any CSP from providing faulty encrypted SLAs.

- **Malicious customer:** A single customer or a group of colluded customers can try to use the *QRES* system to find each CSPs security posture. Each of the colluded entities can specify different requirements and at the end, each customer gets the best matching CSP's secSLA according to her/his requirements. To better illustrate this kind of attack, we show an empirical validation of the proposed system through the secSLA information used in the implementation section; *a secSLA with 50 SLOs, where each SLO is composed of four service levels*. The number of possible SLA combinations is "4^{50}" (i.e., generally x^y, where x is the number of service levels and y is the number of SLOs). Each customer keyword takes on average "0.52" seconds to search over one CSP's encrypted secSLA with 50 SLOs with 1 level each (i.e., we

demonstrate the system evaluation and performance in Section 5.5; as depicted in Figure 5.5). Thus to find all the possible combinations, it takes $2 * 10^{22}$ years to learn a CSP encrypted secSLA. Hence, this type of attack is not feasible in our system. In addition, countermeasures with specific time between sequential queries could be implemented in the either the broker or the CSP.

- **Malicious broker:** A theoretical attack that a broker could perform would be to attempt and alter CSC's keywords. However, such an attack would be immediately detected by the customer as at the end they get to know the security details they 'negotiated' with the Cloud provider.

System Changes: The *QRES* system should prevent malicious customers to ascertain information about each CSP offered services (i.e., by performing several searching iterations, each iteration with different requirements or by performing frequency analysis). To prevent malicious customers from learning information about CSPs SLAs, *QRES* allows the customers to search for their requirements over "anonymous" CSPs encrypted secSLAs. The CSPs anonymity mitigates the malicious customers attacks as customers learn the result of the searching queries and nothing else.

In order for *QRES* to provide anonymity for the CSPs, the auditor should first generate a unique authentication secret for each CSP during their registration phase. This secret then is send to the broker. Every communication between the broker and the CSP runs over onion routing [RSG98] anonymity network to ensure anonymity of CSPs. In addition, the CSP generates an authentication challenge using the hash of a nonce and their unique authentication secrets. The challenge together with the nonce are send to the broker. The broker stores the nonces and the challenges alongside with every CSP's secSLA, but cannot tell the real identities of the CSPs. After finding the best matching CSP, the selected CSP's authentication secret is sent to the auditor by the broker in order to identify the CSP's identity. Next, the auditor sends the selected CSP's identity to B to manage the agreement between both the CSP and the customer.

Verifying Anonymity: In order to verify the anonymity property of this modification of the *QRES* system, we use the ProVerif tool. We model two processes that are replicated an unbounded number of time and executed in parallel. In each process two CSPs participate by sending their tokens. First, each CSP constructs an OR circuit and sends the onion data ($\mathsf{CSP}_1 \leftrightarrow N_1 \leftrightarrow N_2 \leftrightarrow N_3$) and ($\mathsf{CSP}_2 \leftrightarrow N_1 \leftrightarrow N_2 \leftrightarrow N_3$). Then, each of the intermediate nodes (N_1 and N_2) removes one layer of encryption and at the end forwards the onion to N_3. Finally, once the exit node N_3 receives the two onions from the two CSPs, it removes the last layer and sends the messages to B.

In the first process P_1, CSP_1 and CSP_2 sends two tokens t_1 and t_2, respectively. Once the exit node N_3 removes the last onion layer, it sends the message to B on a public channel $(t_1 || t_2)$. In the second process P_2, the two tokens are swapped; such that CSP_1 and CSP_2 sends t_2 and t_1, respectively. Similarly, N_3 publishes the message on a public channel $(t_2 || t_1)$. The CSPs anonymity is preserved if an attacker cannot distinguish between the two messages and hence, cannot learn which token is sent by which CSP. Nodes transfer messages to each other using a public channel.

If the two defined processes are observationally equivalent $P_1(t_1, t_2) \approx P_2(t_2, t_1)$, we say that the attacker cannot distinguish between t_1 and t_2, which means the attacker is unable to distinguish when the message changes. Hence, the attacker is not able to link

two communication streams of the same CSP, and thus cannot learn which message is sent by which CSP.

Theorem 3: The observational equivalence of P_1 and P_2 holds ($P_1 \approx P_2$).

5.7 RELATED WORK

There exists a large number of searchable encryption schemes. For example, fully homomorphic encryption (FHE) [Geno9] and other general functional encryption schemes, such as Garg et al. [GGH+16], can be used to compute functions over encrypted data. Some recent systems such as CryptDB [PRZB11] and Mylar [PSH+14] support secure computation efficiently. However, some of them do not address all of our desired security properties. In [SLPR15], Sherry et al. utilize secure computation on encrypted data for deep packet inspection. Sherry et al. obfuscates the encryption algorithm using Yao garbled circuit [Yao86; LPo9] and oblivious transfer [ALSZ13]. Garbled circuits have been studied over the years: for example, Faust et al. [FRR+10], Huanf et al. [HEKM11], Kreuter et al. [KSS12], Songhori et al. [SHS+15; SZD+16] and Dziembowski et al. [DFS16]. Furthermore, Asharov et al. [ALSZ15; ALSZ17] present an oblivious transfer extension protocol with respect to efficient malicious adversaries.

Additionally, privacy protection algorithms for different applications have been studied. For example, the use of trusted third parties to anonymize consumption data in a smart grid has been proposed by Efthymiou et al. [EK10]. In [BBBK17], Buescher et al. present a study of how many household readings need to be aggregated in order to protect privacy of individual profiles in a smart grid. Furthermore, privacy-preserving evaluation techniques in genetic tests have been increasingly presented over the recent years such as: Mclaren et al. [MRA+16], Dugan et al. [DZ17], Hubaux et al.[HKM17], Sousa et al. [SLH+17] and Raisar et al. [RTH+17]

5.8 SUMMARY

In this chapter we proposed a system called *QRES*. The proposed system enables CSPs to create security SLAs and publish them encrypted. With the help of an intermediate node, customers can find the best matching Cloud provider by contrasting their requirements against the encrypted secSLAs. Our system is utilizing *QeSe* protocol, a searchable encryption scheme protected by secure two party computation. The inputs of every party remain private while the output is only learned by the broker and consequently the customer.

We implement *QRES* and formally verify its security and privacy properties using ProVerif. In our real word tests, using existing standardized SLAs by the latest industrial standard, the system proved to be fast for the required use case. In our measurements, it requires less than 30 seconds to privately search for 50 customer keywords at a CSP.

6 CONCLUSION

Choosing a Cloud provider that satisfies the security requirements of the customer has become challenging. Quantification and evaluation offer powerful tools for choosing between different CSPs. While the initial results of such techniques are promising, tackling the dependency relations that span across customer requirements is still not considered. Most of these methodologies do not account for information about dependencies between services. It is important to provide customers with comprehensive support which enables (a) automated conflict detection and explanation dedicated to the dependent relations, (b) the assessment and benchmarking of CSPs according to customer fuzzy and subjective requirements, and (c) threat analysis for estimating the risks based on the customer requirements.

In this dissertation, we have extended two state-of-the-art security evaluation techniques (namely QHP and fuzzy-QHP) to quantitatively assess the security level provided by Cloud secSLAs. The proposed extensions were designed based on the specifics of secSLAs as defined by state-of-the-art works and standardisation bodies. Specifically, fuzzy-QHP implements advanced security metrics/Cloud secSLA notions e.g., uncertainty. The two techniques rely on a dependency model to detect any conflicts caused by inconsistent customer requirements. Furthermore, explanations of the detected conflicts are generated to identify problematic customer requirements. Using our proposed techniques, we evaluated different CSPs based on varied security specifications with respect to the customer security requirements. Additionally, we addressed different assignments of security levels and weights enabling customers to compare the security levels offered by different CSPs and find the best CSP satisfying their requirements. Both techniques were empirically validated through a couple of case studies using real-world CSP data obtained from the Cloud Security Alliance. Both techniques show the same rankings for customer certain/specific requirements, but with different scales. Only fuzzy-QHP technique is capable of modelling the customer uncertain requirements. Furthermore, a sensitivity analysis is performed to ascertain the security benefits of improving one or more SLOs. The sensitivity analysis helps CSPs to determine which parameter most affects the overall security level (according to the customer's requirements). To identify threats whose impact results in violation of security requirements, a requirement based threat analysis is presented. The presented analysis relies on a novel methodology that can systematically enumerate the dependencies across the service requirements.

Because secSLAs are concrete mechanisms to improve security assurance and transparency in Cloud systems, their quantitative assessment provides a critical element to drive the development of tools aimed to empower customers during the whole Cloud service life-cycle (from procurement to termination). From a CSP perspective, techniques like QHP and fuzzy-QHP trigger on the one hand the adoption of advanced secSLA capabilities (e.g., automation and continuous monitoring), and on the other hand compliance with relevant standards in this field. However, the mechanisms required to validate the enforcement of these specifications throughout the life cycle of the service are still needed. Although many approaches help CSPs to monitor their compliance to the secSLA to take corrective actions in case of violation, existing approaches do not allow customers to man-

age the compliance validation process themselves, consequently, limiting their ability to assess whether contracted security levels are actually provided. In this dissertation, a decentralized customer side monitoring approach to monitor and detect secSLA violations and autonomously compensate customers is proposed. The proposed approach monitors the compliance of Cloud services to the contracted properties in secSLAs. The approach relies on Ethereum blockchain as a decentralized platform to securely store monitoring logs and incorporate secSLAs as smart contracts. The deployed smart contract integrate and aggregate measurable SLOs and check continuously the compliance of Cloud services to contracted SLOs. Furthermore, autonomously compensate the customer on violation. The proposed monitoring approach has been evaluated on commercial IaaS Cloud service. The results have proven our approach suitable for measuring the values of the SLOs and identifying violations of contracted SLO values.

Despite the benefits of adding the security controls to the secSLA, the inclusion of security implementations information is risky as it would enable malicious entities to better orchestrate their attacks and easier discover vulnerabilities. To tackle this problem, a system called *QRES* is proposed. The proposed system enables CSPs to create secSLAs and publish them in an encrypted form. With the help of an intermediate node, customers can find the CSP better matching their security needs by contrasting their requirements against the encrypted secSLAs. Our system is utilizing a searchable encryption scheme protected by secure two-party computation. We implement *QRES* and formally verify its security and privacy properties using ProVerif. In our real-world tests, using existing standardized secSLAs by the latest industrial standard, the system proved to be fast for the required use case. In our measurements, it requires less than 30 seconds to privately search for 50 customer keywords at a CSP. *QRES* is the first step towards providing security assurance and transparency between Cloud customers and CSPs, while at the same time ensures the confidentiality of the CSPs sensitive information.

To summarize, contributions presented in this dissertation were included in six publications [TTLS14; TMT+16; TTLS17; LTTS17; AWT+17; TBL+18].

Part V

APPENDIX

A

FUZZY NOTATIONS

A.1 CRISP AND FUZZY SETS

A crisp set A is defined as a set in the universal set U (which provides the set of all possible values for a variable) $A \subseteq U$. For example, let U be the set of all cars and A is the set of all cars having six cylinders. In a crisp set, an element x is either a member of the set or not. This is defined using a characteristic function called a membership function μ_A such that:

$$\mu_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

On the other hand, a fuzzy set F allow elements to be *partially* in a set. Each element is given a degree of membership in a set. This is done using a membership function μ_F which maps each element in F to real numbers in the interval $[0, 1]$ (where $[0, 1]$ means real numbers between 0 and 1, including 0 and 1) so that $\mu_F : U \Rightarrow [0, 1]$.
For example as introduced in [Men95], let U be the set of all cars. Assume that cars can be viewed as “domestic" or “foreign" where each car can be viewed as “domestic" if it carries the name of a Germany auto manufacturer; otherwise it is “foreign". There is nothing fuzzy about this; however, many of the components for what we consider to be domestic cars are produced outside of Germany. Additionally, some “foreign" cars are manufactured in Germany. Consequently, one could think of the membership functions for domestic and foreign cars looking like μ_D and μ_F depicted in Figure A.1. Thus, if our car has 75% of its parts made in Germany, then $\mu_D(75\%) = 0.9$ and $\mu_F(75\%) = 0.25$ [Men95].

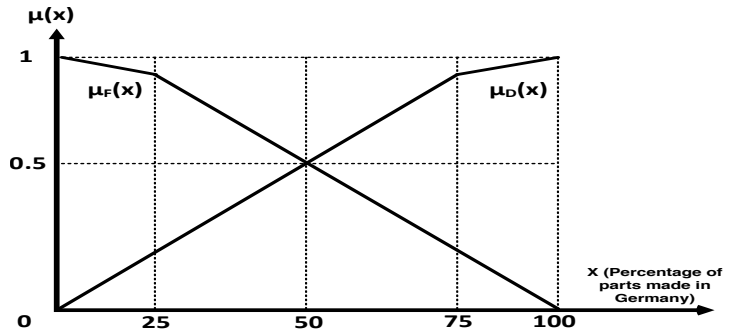

Figure A.1.: Membership functions for domestic and foreign cars, based on the percentage of parts in the car made in Germany [Men95]

It is clear that if one only allowed the extreme membership values of 0 and 1, fuzzy sets would actually be equivalent to crisp sets.

A.2 TRIANGULAR FUZZY NUMBER

A fuzzy number $\tilde{M}$ on the set of real number R ($R \rightarrow [0,1]$) is defined as a TFN if its membership function $\mu_{\tilde{M}}(x)$, whereas x is any positive real number, is equal to (as shown in Figure 2.7):

$$\mu_{\tilde{M}}(x) = \begin{cases} \frac{x-l}{m-l}, & \text{if } l \leqslant x \leqslant m \\ \frac{u-x}{u-m}, & \text{if } m \leqslant x \leqslant u \\ 0, & \text{otherwise} \end{cases}$$

This means a fuzzy set is specified as a TFN if: (i) there exists only one element that the membership function $\mu_{\tilde{M}}(x) = 1$ (at $x = m$) and (ii) $\mu_{\tilde{M}}(x)$ is a continuous function.
For example, let "height" be interpreted as a linguistic variable. It can mean different things to different people and it can be decomposed into the following set of terms: *T(height) = { short, medium, tall}*. Each term in *T(height)* is characterized by a fuzzy set in the universe of discourse $U = [1.2m, 2.2m]$. We might interpret *short* as men height close to 1.4m, *medium* as men height close to 1.7m, and *tall* as men height close to 2m. These terms can be characterized as fuzzy sets whose membership functions are shown in Figure A.2. So for example, $x = 1.52$ resides in the fuzzy set *short* and *medium*, to different degrees of similarity. Clearly, this illustrates the fact that membership functions could be chosen by the user arbitrarily, based on the user's experience or could be designed using optimization procedures (e.g., [HFU92; HSW89]). The number of membership functions is up to the user definition. Greater resolution is achieved by using more membership functions at the price of greater computational complexity. Membership functions don't have to overlap; but, one of the great strengths is that membership functions can be made to overlap [Men95].

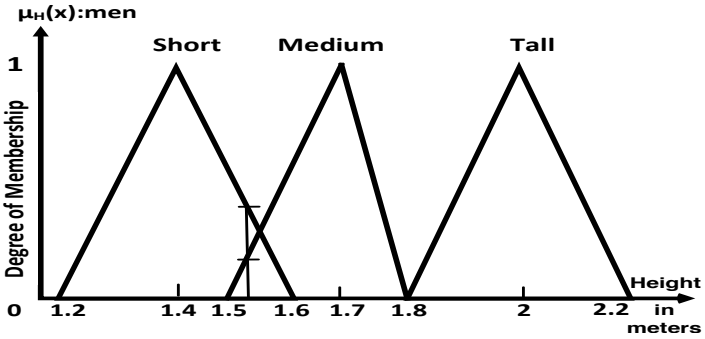

Figure A.2.: Membership functions for *T(height)*.

A.3 OPERATIONS OF TFN - APPROXIMATION OF DIVISION

The approximated value of any two TFNs division $\tilde{A}(/)\tilde{B}$ is equal to $\frac{(l_1, m_1, u_1)}{(l_2, m_2, u_2)}$ such that$(\frac{l_1}{u_2}, \frac{m_1}{m_2}, \frac{u_1}{l_2})$. We explain how to get this approximated value using an example.

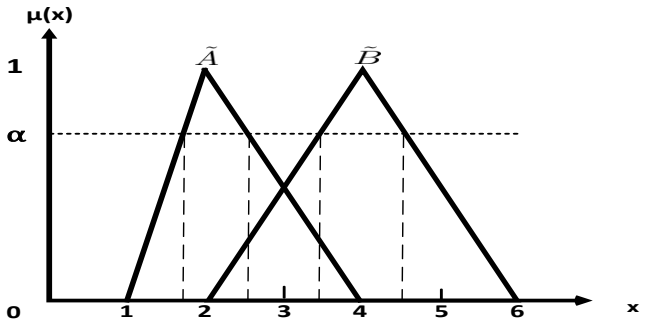

Figure A.3.: Approximation of TFN division example.

Example 1. Consider two TFNs $\tilde{A}$ and $\tilde{B}$ with values $(1,2,4)$ and $(2,4,6)$ respectively as depicted in Figure A.3, such that:

$$\mu_{\tilde{A}}(x) = \begin{cases} \frac{x-1}{2-1}, & \text{if } 1 \leqslant x \leqslant 2 \\ \frac{4-x}{4-2}, & \text{if } 2 \leqslant x \leqslant 4 \\ 0, & \text{otherwise} \end{cases}$$

$$\mu_{\tilde{B}}(x) = \begin{cases} \frac{x-2}{4-2}, & \text{if} 2 \leqslant x \leqslant 4 \\ \frac{6-x}{6-4}, & \text{if} 4 \leqslant x \leqslant 6 \\ 0, & \text{otherwise} \end{cases}$$

First, α-cuts of the two fuzzy numbers are:

$$\begin{aligned} \tilde{A}_\alpha &= [(2-1)\alpha+1, -(4-2)\alpha+4] \\ &= [\alpha+1, -2\alpha+4] \end{aligned}$$

$$\begin{aligned} \tilde{B}_\alpha &= [(4-2)\alpha+2, -(6-4)\alpha+6] \\ &= [2\alpha+2, -2\alpha+6] \end{aligned}$$

For all $\alpha \in [0,1]$, divide interval $\tilde{A}_\alpha$ by $\tilde{B}_\alpha$. Since the element in each interval has positive number, we get:

$$\begin{aligned} \tilde{A}_\alpha(/)\tilde{B}_\alpha = [&\frac{\alpha+1}{2\alpha+2} \wedge \frac{\alpha+1}{-2\alpha+6} \wedge \frac{-2\alpha+4}{2\alpha+2} \wedge \frac{-2\alpha+4}{-2\alpha+6}, \\ &\frac{\alpha+1}{2\alpha+2} \vee \frac{\alpha+1}{-2\alpha+6} \vee \frac{-2\alpha+4}{2\alpha+2} \vee \frac{-2\alpha+4}{-2\alpha+6}] \\ = [&\min(\frac{\alpha+1}{2\alpha+2}, \frac{\alpha+1}{-2\alpha+6}, \frac{-2\alpha+4}{2\alpha+2}, \frac{-2\alpha+4}{-2\alpha+6}), \\ &\max(\frac{\alpha+1}{2\alpha+2}, \frac{\alpha+1}{-2\alpha+6}, \frac{-2\alpha+4}{2\alpha+2}, \frac{-2\alpha+4}{-2\alpha+6})] \end{aligned}$$

Thus,

$$\tilde{A}_\alpha(/)\tilde{B}_\alpha = [\frac{\alpha+1}{-2\alpha+6}, \frac{-2\alpha+4}{2\alpha+2}]$$

$$\text{When} \quad \alpha = 0, \quad \tilde{A}_0(/)\tilde{B}_0 = [\frac{1}{6}, \frac{4}{2}]$$

$$\text{When} \quad \alpha = 1, \quad \tilde{A}_1(/)\tilde{B}_1 = [\frac{(1+1)}{(-2+6)}, \frac{(-2+4)}{(2+2)}]$$

$$= [\frac{2}{4}, \frac{2}{4}]$$

$$= \frac{1}{2}$$

So the approximated value of $\tilde{A}(/)\tilde{B}$ is $(\frac{1}{6}, \frac{1}{2}, 2)$ which is $(\frac{l_1}{u_2}, \frac{m_1}{m_2}, \frac{u_1}{l_2})$. Therefore, any two TFNs division $\tilde{A}(/)\tilde{B}$ equals to:
$\frac{(l_1, m_1, u_1)}{(l_2, m_2, u_2)} = (\frac{l_1}{u_2}, \frac{m_1}{m_2}, \frac{u_1}{l_2})$.

B

OPERATIONAL SEMANTICS

The rules that define the operational semantics of applied pi-calculus and ProVerif are adapted from [Bla09]. The identifiers a, b, c, k and similar ones range over names, and x, y and z range over variables. As detailed in [Bla09], set of symbols is also assumed for constructors and destructors such that f for a constructor and g for a destructor. Constructors are used to build terms. Therefore, the terms are variables, names, and constructor applications of the form $f(M_1, \ldots, M_n)$.

We use the constructors and destructors defined in [Bla09] as an initial step to represent the cryptographic operations as depicted in Figure B.1. We added other different constructors/destructors which are used to define our protocol. Constructors and destructors can be public or private. The public ones can be used by the adversary, which is the case when not stated otherwise. The private ones can be used only by honest participants.

Symmetric enc/dec:
Constructor: encryption of x with the shared secret key k, $senc(x, k)$
Destructor: decryption $sdec(senc(x, k), k) \rightarrow x$
Asymmetric enc/dec:
Constructor: encryption of x with the public key generation from a secret key k, $pk(k)$, $aenc(x, pk(k))$
Destructor: decryption $adec(aenc(x, pk(k)), k) \rightarrow x$
Signatures:
Constructors: signature of x with the secret key k, $sign(x, k)$
Destructors: signature verification using he public key generation from a secret key $k, pk(k), verify(sign(x, k), pk(k)) \rightarrow x$
One-way garbling function:
Constructors: garbling of x with the key $k, garble(x, k)$
Evaluation function:
Constructors: evaluation function of garbling of variables x, y, and z with the key k, $evaluate(garble(x, k), garble(y, k), garble(z, k))$
Commitment:
Constructors: committing x with a fresh nonce n $k, commit(x, n)$

Figure B.1.: Constructors and destructors

The operational semantics used are presented in Figure B.2. A semantic configuration is a pair $\mathcal{E}$,$\mathcal{P}$ where the $\mathcal{E}$ is a finite set of names and $\mathcal{P}$ is a finite multiset of closed processes. The semantics of the calculus is defined by a reduction relation $\rightarrow$ on semantic configurations as shown in Figure B.2. The process $event(M).P$ executes the event $event(M)$ and then executes P. The input process $in(M, x).P$ inputs a message, with x bound to it, on channel M, and executes P. The output process $out(M, N).P$ outputs the message N on the channel M and then executes P. The nil process 0 does nothing. The process $P|Q$ is the parallel composition of P and Q. The replication $!P$ represents an unbounded number of copies of P in parallel. $(new a)P$ creates a new name a and then executes P. The

$$
\begin{array}{ll}
(\mathtt{Nill}) & \mathcal{E}, \mathcal{P} \cup \{0\} \to \mathcal{E}, \mathcal{P} \\
(\mathtt{Repl}) & \mathcal{E}, \mathcal{P} \cup \{!\mathsf{P}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, !\mathsf{P}\} \\
(\mathtt{Par}) & \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\} \\
(\mathtt{Par}) & \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}|\mathsf{Q}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}, \mathsf{Q}\} \\
(\mathtt{New}) & \mathcal{E}, \mathcal{P} \cup \{(\mathit{new}\, a)\mathsf{P}\} \to \mathcal{E} \cup \{a'\}, \mathcal{P} \cup \{\mathsf{P}\{a'/a\}\} \text{ where } a' \notin \mathcal{E} \\
(\mathtt{I/O}) & \mathcal{E}, \mathcal{P} \cup \{\mathtt{out}(c, M).\mathsf{Q}, \mathtt{in}(c, x).\mathsf{P}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}, \mathsf{P}\{M/x\}\} \\
(\mathtt{Cond1}) & \mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\} \text{ if } M = N \\
(\mathtt{Cond2}) & \mathcal{E}, \mathcal{P} \cup \{\text{if } M = N \text{ then } \mathsf{P} \text{ else } \mathsf{Q}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{Q}\} \text{ if } M \neq N \\
(\mathtt{Let}) & \mathcal{E}, \mathcal{P} \cup \{\text{let } x = g(M_1, \ldots, M_n) \text{ in } \mathsf{P} \text{ else } \mathsf{Q}\} \to \mathcal{E}, \mathcal{P} \cup \{\mathsf{P}\{M'/x\}\} \\
 & \text{if } g(M_1, \ldots, M_n) \to M'
\end{array}
$$

Figure B.2.: Operational semantics [BAF08]

conditional if $M = N$ then P else Q executes P if M and N reduce to the same term at runtime; otherwise, it executes Q. Finally, let $x = M$ in P as syntactic for $\mathsf{P}\{M/x\}$ which is the process obtained from P by replacing every occurrence of x with M. As usual, we may omit an else clause when it consists of 0.

B.1 PROTOCOL MODELLING AND PROPERTIES VERIFICATION

In this section we model the *Qese* protocol depicted in Figures 5.3, then verify the result integrity property.

$$
\begin{array}{l}
-\mathsf{P}_B(sk_B, pk_B, m_B) = !\mathtt{in}(c, m).(\mathit{new}\, b) \\
\mathit{event}(e_1(\mathit{commit}(m_B, b))). \\
\mathtt{out}(c, \mathit{commit}(m_B, b)).\mathtt{in}(c, m'').\mathtt{let}((x_B, x_f, x_{CSP}, m_x) = \\
\mathit{adec}(m'', sk_B))\mathtt{in if} m_x = \mathit{commit}(m_B, b)\mathtt{then} \\
\mathit{event}(e_B(\mathit{commit}(m_B, b), x_B, x_f, x_{CSP}, \mathit{evaluate}(x_B, x_f, x_{CSP}))). \\
\mathtt{out}(c, \mathit{evaluate}(x_B, x_f, x_{CSP})) \\
-\mathsf{P}_{CSP}(pk_B, m_f, m_{CSP}) = \mathtt{in}(c, m').\mathtt{let}((y_B = \mathit{garble}(m', k))| \\
(y_f = \mathit{garble}(m_f, k))|(y_{CSP} = \mathit{garble}(m_{CSP}, k)))\mathtt{in} \\
\mathit{event}(e_2(m', y_B, y_f, y_{CSP})). \\
\mathtt{out}(c, (\mathit{senc}((y_B, y_f, y_{CSP}, m'), sk_{CSP})) \\
.\mathtt{in}(c, m''') \\
\mathtt{if} m''' = \mathit{evaluate}(y_B, y_f, y_{CSP})\mathtt{then} \\
\mathit{event}(e_{CSP}(m', y_B, y_f, y_{CSP}, m''')) \\
-\mathsf{P}(\mathit{new} m_f)(\mathit{new} m_{CSP})(\mathit{new} m_B)(\mathit{new} sk_B)\mathtt{let} pk_B = pk_{sk_B}\mathtt{in} \\
\mathtt{out}(c, pk_B).\mathsf{P}_B(sk_B, pk_B, m_B)|\mathsf{P}_{CSP}(pk_B, m_f, m_{CSP})
\end{array}
$$

The channel c is public so that the adversary can send, replay and get any messages sent over it. We use a single public channel and not two or more channels because the adversary could take a message from one channel and relay it on another channel, thus removing any difference between the channels. The process P begins with the creation of the secret and public keys of B, and the creation of messages m_f, m_{CSP}, m_B The public key is output on channel c to model that the adversary has it in its initial knowledge. Then

the protocol itself starts: $\mathsf{P_B}$ represents B, $\mathsf{P_{CSP}}$ represents the CSP. Both principals can run an unbounded number of sessions, so $\mathsf{P_B}$ and $\mathsf{P_{CSP}}$ start with replications.

We consider that B first inputs a message containing the encrypted tokens and then starts the protocol run by choosing a nonce b, and executing the event $e_1(\mathit{commit}(m_B, b))$, where m_B is initially added to the B knowledge. Intuitively, this event records that B sent $Message_1$ of the protocol. Event e_1 is placed before the actual output of $Message_1$; this is necessary for the desired correspondences to hold: if event e_1 followed the output of $Message_1$, we would not be able to prove that event e_1 must have been executed, even though $Message_1$ must have been sent, because $Message_1$ could be sent without executing event e_1, as stated in [Bla09]. The situation is similar for events e_2, e_B and e_{CSP}.

Next, B receives the garbling of CSP's inputs as well as the garbling of the committed messages encrypted with its public key. B decrypts the message using its secret key sk_B. If decryption succeeds B checks if the message has the right form using the pattern-matching construct $\mathtt{let}((x_B, x_f, x_{CSP}, = m_B) = \mathit{adec}(m'', sk_B))\mathtt{in}$. Then B executes the event $e_B(\mathit{commit}(m_B, b), x_B, x_f, x_{CSP},$
$\mathit{evaluate}(x_B, x_f, x_{CSP}))$, to record that it has received $Message_2$ and sent $Message_3$ of the protocol. Finally, B sends the last message of the protocol $\mathit{evaluate}(x_B, x_f, x_{CSP})$.

After sending this message, B executes some actions needed only for specifying properties of the protocol. When the received message $m_x = \mathit{commit}(m_B, b)$, that is, when the session is between B and CSP, B executes the event $e_B(\mathit{commit}(m_B, b), x_B, x_f, x_{CSP},$
$\mathit{evaluate}(x_B, x_f, x_{CSP}))$, to record that B ended a session of the protocol, with the participant (CSP), which is authenticated using the authentication key. B also outputs the evaluation function output $\mathit{evaluate}(x_B, x_f, x_{CSP})$.

The process $\mathsf{P_{CSP}}$ proceeds similarly: it executes the protocol, with the additional event $e_2(m', y_B, y_f, y_{CSP})$ to record that $Message_1$ has been received and $Message_2$ has been sent by CSP, in a session with the participant of public key pk_B and the received message m'. After finishing the protocol itself, when $m''' = \mathit{evaluate}(y_B, y_f, y_{CSP})$, that is, when the session is between B and CSP, $\mathsf{P_{CSP}}$ executes the event $e_{CSP}(m', y_B, y_f, y_{CSP}, m''')$, to record that CSP finished the protocol, and outputs m'''.

The events will be used in order to formalize result integrity. For example, we formalize that, if CSP ends a session of the protocol $e_{CSP}(m', y_B, y_f, y_{CSP}, m''')$, then (a) B has started a session of the protocol by committing m_B with the nonce n_B, and (b) CSP outputs the evaluation function $\mathit{evaluate}(y_B, y_f, y_{CSP})$. Furthermore, B ends a session of the protocol, then (a) CSP has already garbled the B's committed input message, and (b) B outputs the evaluation function $\mathit{evaluate}(x_B, x_f, x_{CSP})$.

Next, we formally define the correspondences in order to verify the result integrity property. We prove correspondences in the form of *if an event e has been executed, then events* $e_1, \ldots, e_m$ *have been executed*. These events may include arguments, which allows one to relate the values of variables at the various events. We can prove that each execution of e corresponds to a distinct execution of some events, and that the events have been executed in a certain order. We assume that the protocol is executed in the presence of an adversary that can listen to all messages, compute, and send all messages it has, following the so-called Dolev-Yao model [DY83]. Thus, an adversary can be represented by any process that has a set of public names $\mathtt{Init}$ in its initial knowledge and that does not contain events.

As presented in system model, the correspondence event $e_{CSP}(x_1,$
$x_2, x_3, x_4, x_5) \rightsquigarrow e_1(x_1) \wedge e_2(x_1, x_2, x_3, x_4) \wedge e_B(x_1, x_2, x_3, x_4, x_5)$ means that, if the event

$e_{\mathsf{CSP}}(x_1, x_2, x_3, x_4, x_5)$ has been executed, then the events $e_1(x_1), e_2(x_1, x_2, x_3, x_4)$ and $e_{\mathsf{B}}(x_1, x_2, x_3, x_4, x_5)$ have been executed, with the same value of the arguments x_1, x_2, x_3, x_4, x_5.

BIBLIOGRAPHY

[ABB+05] Alessandro Armando, David Basin, Yohan Boichut, et al. "The AVISPA tool for the automated validation of internet security protocols and applications." In: *Proc. of CAV*. 2005, pp. 281–285.

[AF01] Martín Abadi and Cédric Fournet. "Mobile values, new names, and secure communication." In: *Sigplan Notices* 36.3 (2001), pp. 104–115.

[AGI11] Mohamed Almorsy, John Grundy, and Amani Ibrahim. "Collaboration-Based Cloud Computing Security Management Framework." In: *Proc. of Cloud Computing*. 2011, pp. 364–371.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. "More efficient oblivious transfer and extensions for faster secure computation." In: *Proc. of CCS*. 2013, pp. 535–548.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. "More efficient oblivious transfer extensions with security for malicious adversaries." In: *Proc. of EUROCRYPT*. Springer. 2015, pp. 673–701.

[ALSZ17] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. "More efficient oblivious transfer extensions." In: *Journal of Cryptology* 30.3 (2017), pp. 805–858.

[AM13] Hamzeh Alabool and Ahmad Mahmood. "Trust-based service selection in public cloud computing using fuzzy modified VIKOR method." In: *Australian Journal of Basic and Applied Sciences* (2013), pp. 211–220.

[Ama] *Amazon CloudWatch*. https://aws.amazon.com/cloudwatch/. [Online; accessed 25-Feb-2018].

[AWT+17] Soha Alboghdady, Stefan Winter, Ahmed Taha, et al. "C'Mon: Monitoring the Compliance of Cloud Services to Contracted Properties." In: *Proc. of ARES*. 2017, p. 36.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated verification of selected equivalences for security protocols." In: *Journal of Logic and Algebraic Programming* 75.1 (2008), pp. 3–51.

[Bar11] A. Barth. *RFC 6265 HTTP State Management Mechanism*. [Online; accessed 22-March-2018]. 2011. URL: `https://tools.ietf.org/html/rfc6265\#section-4.1.2.5`.

[BBBK17] Niklas Buescher, Spyros Boukoros, Stefan Bauregger, and Stefan Katzenbeisser. "Two Is Not Enough: Privacy Assessment of Aggregation Schemes in Smart Metering." In: *Privacy Enhancing Technologies* 2017.4 (2017), pp. 198–214.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. "Deterministic and efficiently searchable encryption." In: *Proc. of CRYPTO*. 2007, pp. 535–552.

[BCOP04] Dan Boneh, Giovanni Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. "Public key encryption with keyword search." In: *Proc. of EUROCRYPT*. 2004, pp. 506–522.

[Bla01] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules." In: *Proc. of CSFW*. 2001, pp. 82–96.

[Bla02] Bruno Blanchet. "From secrecy to authenticity in security protocols." In: *Proc. of SAS*. 2002, pp. 342–359.

[Bla09] Bruno Blanchet. "Automatic verification of correspondences for security protocols." In: *Journal of Computer Security* 17.4 (2009), pp. 363–434.

[Blo] *blockchain network types*. [Online; accessed 27-Feb-2018]. URL: `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/`.

[BNP+11] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, et al. "AmazonIA: when elasticity snaps back." In: *Proc. of CCS*. 2011, pp. 389–400.

[Bre11] Michael Brenner. *Yao's Garbled Circuits implementation*. 2011. URL: `https://github.com/hcrypt-project/yao`.

[Bro01] Tyson Browning. "Applying the design structure matrix to system decomposition and integration problems: a review and new directions." In: *Transaction on Engg. Management* 48.3 (2001), pp. 292–306.

[Cha96] Da Chang. "Applications of the extent analysis method on fuzzy AHP." In: *European journal of operational research* 95.3 (1996), pp. 649–655.

[Cis] *The CIS Security Metrics v1.1.0*. Tech. rep. National Institute of Standards and Technology, 2010. URL: `https://benchmarks.cisecurity.org/tools2/metrics/CIS\_Security\_Metrics\_v1.1.0.pdf`.

[CLN09] Cas Cremers, Pascal Lafourcade, and Philippe Nadeau. "Comparing state spaces in automatic security protocol analysis." In: *Proc. of Formal to Practical Security*. 2009, pp. 70–94.

[CMMR06] Valentina Casola, Antonino Mazzeo, Nicola Mazzocca, and Massimiliano Rak. "A SLA evaluation methodology in Service Oriented Architectures." In: *Quality of Protection* (2006), pp. 119 –130.

[Cre08] Cas Cremers. "The Scyther Tool: Verification, falsification, and analysis of security protocols." In: *Proc. of CAV*. 2008, pp. 414–418.

[Csi] *Cloud Service Level Agreement Standardisation Guidelines*. Tech. rep. C-SIG SLA 2014. European Commission, C-SIG SLA, 2014.

[CWL10] Shirlei Chaves, Carlos Westphall, and Flavio Lamin. "SLA perspective in security management for cloud computing." In: *Proc. of Networking and Services*. 2010, pp. 212–217.

[CYZ+12] Chunqing Chen, Shixing Yan, Guopeng Zhao, et al. "A systematic framework enabling automatic conflict detection and explanation in cloud service selection for enterprises." In: *Proc. of Cloud Computing* (2012), pp. 883–890.

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. "Private circuits III: Hardware Trojan-Resilience via testing amplification." In: *Proc. of CCS*. 2016, pp. 142–153.

[DIR02] Nicole Dunlop, Jadwiga Indulska, and Kerry Raymond. "Dynamic conflict detection in policy-based management systems." In: *Proc. of the Enterprise Distributed Object Computing Conference* (2002), pp. 15–26.

[DVL15] Andrea Dalle Vacche and Stefano Kewan Lee. *Zabbix network monitoring essentials*. Packt Publishing Ltd, 2015.

[DY83] Danny Dolev and Andrew Yao. "On the security of public key protocols." In: *Transactions on information theory* 29.2 (1983), pp. 198–208.

[DZ17] Tamara Dugan and Xukai Zou. "Privacy-preserving evaluation techniques and their application in genetic tests." In: *Smart Health* 1 (2017), pp. 2–17.

[EK01] Christian Ensel and Alexander Keller. "Managing application service dependencies with xml and the resource description framework." In: *Proc. of the Integrated Network Management Proceedings* (2001), pp. 661–674.

[EK10] Costas Efthymiou and Georgios Kalogridis. "Smart grid privacy via anonymization of smart metering data." In: *Proc. of SmartGridComm*. 2010, pp. 238–243.

[Ene] *Bitcoin Mining Now Consuming More Electricity*. [Online; accessed 27-Feb-2018]. URL: `https://powercompare.co.uk/bitcoin`.

[Eth] *Ethereum White Paper*. [Online; accessed 27-Feb-2018]. URL: `https://github.com/ethereum/wiki/wiki/White-Paper`.

[Evm] *Ethereum Virtual Machine*. [Online; accessed 22-Feb-2018]. URL: `http://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html\#index-6`.

[FMP+17] Md Sadek Ferdous, Andrea Margheri, Federica Paci, et al. "Decentralised runtime monitoring for access control systems in cloud federations." In: *Proc. of ICDCS*. 2017, pp. 2632–2633.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, et al. "Protecting circuits from leakage: the computationally-bounded and noisy cases." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2010, pp. 135–156.

[FY07] Ganna Frankova and Artsiom Yautsiukhin. "Service and protection level agreements for business processes." In: *Proc. of ICSOC*. 2007, pp. 38 –43.

[GAB+17] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, et al. "Blockchain-based database to ensure data integrity in cloud computing environments." In: *ITA Security* (2017).

[GE91] David Gebala and Steven Eppinger. "Methods for Analyzing Design Procedures." In: *Proc. of Design Theory and Methodology* (1991), pp. 227–233.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *Proc. of STOC*. 2009, pp. 169–178.

[Get] *GO Implementation of Ethereum Node*. [Online; accessed 21-Feb-2018]. URL: `https://github.com/ethereum/go-ethereum/wiki/geth`.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, et al. "Candidate indistinguishability obfuscation and functional encryption for all circuits." In: *SICOMP* 45.3 (2016), pp. 882–929.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[GVB13] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. "A framework for comparing and ranking cloud services." In: *Journal of Future Generation Computer Systems* (2013), pp. 1012–1023.

[HBS10] Irfan Haq, Ivona Brandic, and Erich Schikuta. "Sla validation in layered cloud infrastructures." In: *Proc. of GECON*. 2010, pp. 153–164.

[HD12] Giles Hogben and Marnix Dekker. *Procure Secure: A guide to monitoring of security service levels in cloud contracts*. Tech. rep. 2012. URL: https://www.enisa.europa.eu/publications/procure-secure-a-guide-to-monitoring-of-security-service-levels-in-cloud-contracts.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. "Faster Secure Two-Party Computation Using Garbled Circuits." In: *USENIX Security Symposium*. Vol. 201. 1. 2011, pp. 331–335.

[Hen99] Ronda Henning. "Security SLAs: quantifiable security for the enterprise?" In: *Proc. of Workshop New Sec Paradigms*. 1999, pp. 54–60.

[HFU92] Shin Horikawa, Takeshi Furuhashi, and Yoshiki Uchikawa. "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm." In: *IEEE transactions on Neural Networks* 3.5 (1992), pp. 801–806.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. "Efficient secure two-party computation using symmetric cut-and-choose." In: *Proc. of CRYPTO*. 2013, pp. 18–35.

[HKM17] Jean-Pierre Hubaux, Stefan Katzenbeisser, and Bradley Malin. "Genomic Data Privacy and Security: Where We Stand and Where We Are Heading." In: *IEEE Security & Privacy* 5 (2017), pp. 10–12.

[HLOS06] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. "Threat modeling-uncover security design flaws using the stride approach." In: *MSDN Magazine-Louisville* (2006), pp. 68–75.

[HRM11] Sheikh Habib, Sebastian Ries, and Max Muhlhauser. "Towards a trust management system for cloud computing." In: *Proc. of TrustCom* (2011), pp. 933–939.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

[IL01] Cynthia Irvine and Timothy Levin. "Quality of security service." In: *Proc. of workshop on New security paradigms*. 2001, pp. 91–99.

[JW98] George Jelen and Jeffrey Williams. "A practical approach to measuring assurance." In: *Pro. of Computer Security Applications*. 1998, pp. 333–343.

[KH11] Golam Kabir and M Hasin. "Comparative analysis Of AHP and fuzzy AHP models for multicriteria inventory classification." In: *Journal of Fuzzy Logic Systems* (2011), pp. 1–16.

[KMY11] Leanid Krautsevich, Fabio Martinelli, and Artsiom Yautsiukhin. "A General Method for Assessment of Security in Complex Services." In: *Proc. of European Conference on a Service-Based Internet*. 2011, pp. 153–164.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. "Dynamic searchable symmetric encryption." In: *Proc. of CCS*. 2012, pp. 965–976.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. "Billion-Gate Secure Computation with Malicious Adversaries." In: *USENIX Security Symposium*. Vol. 12. 2012, pp. 285–300.

[Lee18] Jong-Hyouk Lee. "BIDaaS: Blockchain based ID as a Service." In: *IEEE Access* (2018).

[LF08] André Ludwig and Bogdan Franczyk. "COSMA–an approach for managing sLAs in composite services." In: *Proc. of Service-Oriented Computing* (2008), pp. 626–632.

[LGLS12] Jesus Luna Garcia, Robert Langenberg, and Neeraj Suri. "Benchmarking cloud security level agreements using quantitative policy trees." In: (2012), pp. 103–112.

[Lin05] Stefan Lindskog. *Modeling and tuning security from a quality of service perspective*. Chalmers University of Technology, 2005.

[LKL13] Duo Liu, Utkarsh Kanabar, and Chung Lung. "A light weight SLA management infrastructure for cloud computing." In: *Proc. of CCECE*. 2013, pp. 1–4.

[LP07] Yehuda Lindell and Benny Pinkas. "An efficient protocol for secure two-party computation in the presence of malicious adversaries." In: *Proc. of EUROCRYPT*. 2007, pp. 52–78.

[LP09] Yehuda Lindell and Benny Pinkas. "A proof of security of Yao's protocol for two-party computation." In: *Journal of Cryptology* 22.2 (2009), pp. 161–188.

[LTTS17] Jesus Luna, Ahmed Taha, Ruben Trapero, and Neeraj Suri. "Quantitative Reasoning about Cloud Security Using Service Level Agreements." In: *IEEE Transactions on Cloud Computing* 5.3 (2017), pp. 457–471.

[LYC+17] Bin Liu, Xiao Liang Yu, Shiping Chen, et al. "Blockchain Based Data Integrity Service Framework for IoT Data." In: *Proc. of Web Services*. 2017, pp. 468–475.

[LYKZ10] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. "CloudCmp: comparing public cloud providers." In: *Proc. of SIGCOMM* (2010), pp. 1–14.

[Men95] Jerry Mendel. "Fuzzy logic systems for engineering: a tutorial." In: *Proc. of IEEE* 83.3 (1995), pp. 345–377.

[Mes06] Mandy Messenger. *Cyber-security: Why would I tell you?* Tech. rep. 3. Research briefing report by Mandy Messenger, 2006. URL: `https://archive.nyu.edu/bitstream/2451/15007/2/Infosec_ISR_Messenger.pdf`.

[MFYS17] Andrea Margheri, Md Sadek Ferdous, Mu Yang, and Vladimiro Sassone. "A distributed infrastructure for democratic cloud federations." In: *Proc. of CLOUD*. 2017, pp. 688–691.

[MJSCG04] Carlos Molina-Jimenez, Santosh Shrivastava, Jon Crowcroft, and Panos Gevros. "On the monitoring of contractual service level agreements." In: *Proc. of Electronic Contracting*. 2004, pp. 1–8.

[MKS13] Saif Malik, Samee Khan, and Sudarshan Srinivasan. "Modeling and analysis of state-of-the-art VM-based cloud management platforms." In: *IEEE Transactions on Cloud Computing* 1.1 (2013), pp. 1–1.

[MM87] David Marca and Clement McGowan. "SADT: structured analysis and design technique." In: *McGraw-Hill* (1987).

[MMW+15] Suryadipta Majumdar, Taous Madi, Yushun Wang, et al. "Security Compliance Auditing of Identity and Access Management in the Cloud: Application to OpenStack." In: *Proc. of CloudCom*. 2015, pp. 58–65.

[MRA+16] Paul McLaren, Jean Raisaro, Manel Aouri, et al. "Privacy-preserving genomic testing in the clinic: a model using HIV treatment." In: *Genetics in medicine* 18.8 (2016), p. 814.

[MTS16] Salman Manzoor, Ahmed Taha, and Neeraj Suri. "Trust Validation of Cloud IaaS: A Customer-centric Approach." In: *Proc. of TrustCom*. 2016, pp. 97–104.

[MTT+16] Jolanda Modic, Ruben Trapero, Ahmed Taha, et al. "Novel efficient techniques for real-time cloud security assessment." In: *Computers & Security* 62 (2016), pp. 1–18.

[Nag] *Nagios Monitoring Tool*. https://www.nagios.org/. [Online; accessed 25-Feb-2018].

[Nak08] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system." In: (2008).

[NG14] David Nunez and Carmen Gago. *D:C-5.2 Validation of the Accountability Metrics*. Tech. rep. Accountability For Cloud and Other Future Internet Services (A4Cloud), 2014. URL: http://www.cloudaccountability.eu/sites/default/files/D35.2%20Validation%20of%20the%20accountability%20metrics.pdf.

[NP99] Moni Naor and Benny Pinkas. "Oblivious transfer with adaptive queries." In: *Proc. of CRYPTO*. 1999, pp. 573–590.

[NS11] Talal Noor and Quan Sheng. "Trust as a service: a framework for trust management in cloud environments." In: *Proc. of Web Information System Engineering* (2011), pp. 314–321.

[Ora] *Oracles in Blockchain Context*. https://blockchainhub.net/blockchain-oracles/. [Online; accessed 21-Feb-2018].

[PBSL13] Diego Perez-Botero, Jakub Szefer, and Ruby Lee. "Characterizing hypervisor vulnerabilities in cloud computing servers." In: *Proc. of Security in cloud computing*. 2013, pp. 3–10.

[PHK+13] Alain Pannetrat, Giles Hogben, Spyros Katopodis, et al. *D2.1 Security-Aware SLA Specification Language and Cloud Security Dependency model*. Tech. rep. 2013.

[Pro14] SPECS Project. *Report on requirements for Cloud SLA negotiation - Final*. Tech. rep. Deliverable 2.1.2. 2014. URL: http://www.specs-project.eu/publications/public-deliverables/d2-1-2/.

[Pro17] ESCUDO Project. *Report on Requirement-Based Threat Analysis*. Tech. rep. Deliverable 2.4. 2017. URL: http://www.escudocloud.eu/public-del/D2.4.pdf.

[PRZB11] Raluca Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. "CryptDB: protecting confidentiality with encrypted query processing." In: *Proc. of SOSP*. 2011, pp. 85–100.

[PSH+14] Raluca Popa, Emily Stark, Jonas Helfer, et al. "Building Web Applications on Top of Encrypted Data Using Mylar." In: *Proc. of NSDI*. 2014, pp. 157–172.

[PSK08] Witold Pedrycz, Andrzej Skowron, and Vladik Kreinovich. *Handbook of granular computing*. John Wiley & Sons, 2008.

[Rab05] Michael Rabin. "How To Exchange Secrets with Oblivious Transfer." In: *Proc. of IACR Cryptology ePrint Archive*. Vol. 2005. 2005, p. 187.

[Rin] *Rinkeby - Ethereum Testnet*. URL: https://rinkeby.etherscan.io/.

[Rop] *Ropsten- Ethereum Testnet*. URL: https://ropsten.etherscan.io/.

[Ros77] Douglas Ross. "Structured analysis (SA): A language for communicating ideas." In: *Software Engineering* 1 (1977), pp. 16–34.

[RSG98] Michael Reed, Paul Syverson, and David Goldschlag. "Anonymous connections and onion routing." In: *Selected areas in Communications* 16.4 (1998), pp. 482–494.

[RTH+17] Jean Raisaro, Carmela Troncoso, Mathias Humbert, et al. *GenoShare: Supporting Privacy-Informed Decisions for Sharing Exact Genomic Data*. Tech. rep. EPFL infoscience, 2017.

[RVEE+11] Massimiliano Rak, Salvatore Venticinque, Gorka Echevarria, Gorka Esnal, et al. "Cloud application monitoring: The mosaic approach." In: *Proc. of CloudCom*. 2011, pp. 758–763.

[RWQ+08] Omer Rana, Martijn Warnier, Thomas Quillinan, et al. "Managing violations in service level agreements." In: *Proc. of Grid Middleware and Services*. Springer, 2008, pp. 349–358.

[Saa90] Thomas Saaty. "How to make a decision: the analytic hierarchy process." In: *European journal of operational research* 48.1 (1990), pp. 9–26.

[SBHD17] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. "Towards Blockchain-based Auditable Storage and Sharing of IoT Data." In: *Proc. of Cloud Computing Security Workshop*. 2017, pp. 45–50.

[SHS+15] Ebrahim Songhori, Siam Hussain, Ahmad-Reza Sadeghi, et al. "Tinygarble: Highly compressed and scalable sequential garbled circuits." In: *Proc. of Security and Privacy*. 2015, pp. 411–428.

[SJSJ05] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. "Using dependency models to manage complex software architecture." In: *Sigplan Notices* 40.10 (2005), pp. 167–176.

[SLH+17] João Sá Sousa, Cédric Lefebvre, Zhicong Huang, et al. "Efficient and secure outsourcing of genomic data storage." In: *BMC medical genomics* 10.2 (2017), p. 46.

[SLPR15] Justine Sherry, Chang Lan, Raluca Popa, and Sylvia Ratnasamy. "Blindbox: Deep packet inspection over encrypted traffic." In: *In SIGCOMM* 45.4 (2015), pp. 213–226.

[Sma] *Smart-Contracts*. [Online; accessed 22-Feb-2018]. URL: http://ethdocs.org/en/latest/contracts-and-transactions/contracts.html.

[SP12] Jane Siegel and Jeff Perdue. "Cloud services measures for global use: the Service Measurement Index." In: *Proc. of SRII Conf* (2012), pp. 411–415.

[SSNM16] Francesco Paolo Schiavo, Vladimiro Sassone, Luca Nicoletti, and Andrea Margheri. "Faas: Federation-as-a-service." In: *arXiv preprint arXiv:1612.03937* (2016).

[Sta] *State of The Dapps*. https://www.stateofthedapps.com/. [Online; accessed 27-Feb-2018].

[Sta13] International Organization for Standardization. *Information Technology, Security Techniques, Code of Practice for Information Security Management*. Tech. rep. ISO/IEC 27002:2013. 2013.

[Ste81] Donald Steward. "The design structure system: a method for managing the design of complex systems." In: *Trans. on Engg. Management* 3 (1981), pp. 71–74.

[Sup+12] M Supriya et al. "Estimating trust value for cloud service providers using fuzzy logic." In: *International Journal of Computer Applications* (2012), pp. 28–34.

[SWP00] Xiaoding Song, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data." In: *Proc. of S&P*. 2000, pp. 44–55.

[SZD+16] Ebrahim M Songhori, Shaza Zeitouni, Ghada Dessouky, et al. "GarbledCPU: a MIPS processor for secure computation in hardware." In: *Proc. of Annual Design Automation Conference*. 2016, p. 73.

[TBL+18] Ahmed Taha, Spyros Boukoros, Jesus Luna, et al. "QRES: Quantitative Reasoning on Encrypted Security SLAs." In: *Privacy Enhancing Technologies* (2018). [Submitted].

[TMS+17] Ruben Trapero, Jolanda Modic, Miha Stopar, et al. "A novel approach to manage cloud security SLA incidents." In: *Future Generation Computer Systems* 72 (2017), pp. 193–205.

[TMS17] Ahmed Taha, Salman Manzoor, and Neeraj Suri. "SLA-Based Service Selection for Multi-Cloud Environments." In: *Proc. of Edge Computing*. 2017, pp. 65–72.

[TMT+16] Ahmed Taha, Patrick Metzler, Ruben Trapero, et al. "Identifying and Utilizing Dependencies Across Cloud Security Services." In: *Proc. of AsiaCCS*. 2016, pp. 329–340.

[TSM+12] Hsin-Yi Tsai, Melanie Siebenhaar, Andre Miede, et al. "Threat as a service?: Virtualization's impact on cloud security." In: *IT professional* 14.1 (2012), pp. 32–37.

[TTLS14] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. "Ahp-based quantitative approach for assessing and comparing cloud security." In: *Proc. of TrustCom*. 2014, pp. 284–291.

[TTLS17] Ahmed Taha, Ruben Trapero, Jesus Luna, and Neeraj Suri. "A Framework for Ranking Cloud Security Services." In: *Proc. of IEEE Services Computing*. 2017, pp. 322–329.

[UA14] Kazi Ullah and Abu Ahmed. "Demo paper: Automatic provisioning, deploy and monitoring of virtual machines based on security service level agreement in the cloud." In: *Proc. of CCGrid*. 2014, pp. 536–537.

[UAY13] Kazi Ullah, Abu Ahmed, and Jukka Ylitalo. "Towards building an automated security compliance tool for the cloud." In: *Proc. of TrustCom*. 2013, pp. 1587–1593.

[Wan09] Ping Wang. "QoS-aware web services selection with intuitionistic fuzzy set under consumer's vague perception." In: *Expert Systems with Applications* 36.3 (2009), pp. 4460–4466.

[WL77] Jerome Wiest and Ferdinand Levy. "A management guide to PERT/CPM." In: *Prentice-Hall* (1977).

[WLK+12] Ping Wang, Wen-Hui Lin, Pu-Tsun Kuo, et al. "Threat risk analysis for cloud security based on Attack-Defense Trees." In: *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*. Vol. 1. 2012, pp. 106–111.

[Woo14] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger." In: *Ethereum Project Yellow Paper* 151 (2014), pp. 1–32.

[WS09] Matthias Winkler and Alexander Schill. "Towards Dependency Management in Service Compositions." In: *Proc. of e-Business* (2009), pp. 79–84.

[WSS10] Matthias Winkler, Thomas Springer, and Alexander Schill. "Automating composite SLA management tasks by exploiting service dependency information." In: *Proc. of Web Services* (2010), pp. 59–66.

[Yao86] Andrew Yao. "How to generate and exchange secrets." In: *Proc. of Symposium on IEEE*. 1986, pp. 162–167.

[ZC73] Milan Zeleny and James Cochrane. "Multiple criteria decision making." In: *University of South Carolina Press* (1973).

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

[ZLLL14] Zhenling Zhang, Lejian Liao, Hai Liu, and Guoqiang Li. "Policy-based adaptive service level agreement management for cloud services." In: *Proc. of ICSESS*. 2014, pp. 496–499.

[Ama12] Amazon Web Services, Inc. *Amazon DynamoDB*. [Online; accessed 15-December-2017]. 2012. URL: https://aws.amazon.com/dynamodb/.

[Ama17] Amazon. *Amazon web services*. http://aws.amazon.com/ec2/. 2017. (Visited on 03/17/2017).

[Clo17a] Cloud Security Alliance. In: *Cloud Controls Matrix v3.0.1* (2017). URL: https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1/.

[Clo17b] Cloud Security Alliance. *The Consensus Assessments Initiative Questionnaire v3.0.1*. 2017. URL: https://cloudsecurityalliance.org/download/consensus-assessments-initiative-questionnaire-v3-0-1/.

[Clo17c] Cloud Security Alliance. "The Open Certification Framework." In: (2017). URL: https://cloudsecurityalliance.org/group/open-certification/#_overview.

[Clo17d] Cloud Security Alliance. "The Security, Trust & Assurance Registry (STAR)." In: (2017). URL: https://cloudsecurityalliance.org/star/#_overview.

[Int14] International Organization for Standardization. *Information Technology - Cloud Computing - Service Level Agreement (SLA) Framework and Terminology*. Tech. rep. ISO/IEC 19086. 2014.

[NIS08] NIST Cloud Computing Reference Architecture and Taxonomy Working Group. "Performance and Measurements Guide for Information Technology." In: *NIST 800-55 Revision 1* (2008). URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-55r1.pdf.

[Nat13] National Institute of Standards and Technology. *Security Controls for Federal Information Systems*. Tech. rep. NIST SP-800-53. 2013.

[QRE17] QRES. *Quantitative Reasoning on Encrypted Security SLAs*. 2017. URL: https://github.com/Anonymous12-34/QRESProject.