IaaS-cloud Security Enhancement: An Intelligent Attribute-Based Access Control Model and Implementation

by

Shadha Mohamed Sulaiyam ALAmri

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

18th September 2017

Abstract

The cloud computing paradigm introduces an efficient utilisation of huge computing resources by multiple users with minimal expense and deployment effort compared to traditional computing facilities. Although cloud computing has incredible benefits, some governments and enterprises remain hesitant to transfer their computing technology to the cloud as a consequence of the associated security challenges. Security is, therefore, a significant factor in cloud computing adoption. Cloud services consist of three layers: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Cloud computing services are accessed through network connections and utilised by multi-users who can share the resources through virtualisation technology. Accordingly, an efficient access control system is crucial to prevent unauthorised access.

This thesis mainly investigates the IaaS security enhancement from an access control point of view. IaaS requires an access control model that can cope with its dynamic and scalable features. Attribute-based Access Control (ABAC) is identified as the most appropriate model that can support IaaS features. However, ABAC does encounter challenges regarding its conceptual characteristics and formal specifications.

Thus, a novel intelligent attribute-based access control model known as $ABAC_{sh}$ is proposed in this thesis for IaaS cloud. $ABAC_{sh}$ is an enhanced attribute-based access control that supports a context-aware mechanism and the Separation of Duty (SoD) security principle. The intelligent framework for $ABAC_{sh}$ is presented based on a knowledge agent, as logic is used to infer an access decision. Hence, there is a requirement to formulate a suitable logic that is able to specify, verify and reason ABAC models. A formal logic known as Access Control Logic-Deontic Logic (ACL-DL) is further proposed in this thesis. ACL-DL involves a formal language definition and a Kripke-structure model definition. ACL-DL logic is used to formally specify $ABAC_{sh}$ properties and to logically reason about $ABAC_{sh}$ access requests.

To present the feasibility of the proposed access control model in an IaaS platform, a demonstration of $ABAC_{sh}$ prototype is studied in an open-source IaaS-platform known as OpenStack; proposing an enforcement framework that is designed based on agent architecture. Furthermore, an extended Policy Decision Point (PDP) is implemented based on a logical inference algorithm known as forward chaining.

Formal verification proves that the ACL-DL satisfaction problem is PSPACE solvable, and its model-checking problem is PTIME solvable. Evaluation of the logic complexity profile illustrates that ACL-DL logic is computationally determ-

inable, in contrast with classical logic categories. Evaluation of the $ABAC_{sh}$ computation complexity and access control quality metrics reveals superior results in comparison with two existing ABAC models. The experimental results in an OpenStack testbed deployment demonstrate the feasibility of the $ABAC_{sh}$ approach and promise an enhanced and secure access control model for IaaS cloud.

Keywords

IaaS; Cloud Computing; Security; Attribute based Access Control; Modal logic; Deontic Logic; Artificial intelligence; OpenStack

Acknowledgements

To Allah, who provides me with unlimited strength and guidance, and who was beside me all the time when no one was able to understand me and when the PhD stress reached its peak. To you, my lord, my profound gratitude and thanks.

To my husband, Ahmed AL-Harbi, and my two sons, Emran and Hamzah, who accompany me on my challenging and prolonged journey.

To my father, Mohamed, and my mother, Sharifa, the most precious gifts I have in this life, who have instilled the love of learning in me and my nine siblings, Ahmed, Faisal, Shaima, Hamed, Husam, Omar, Bader, Mustafa and Shahed. We are a big family with a big heart and strong bonds. My thanks to my lovely family.

My further acknowledgement and thanks to my beloved country Oman for supporting me in my higher education since my BSc (Sultan Qaboos University, 2005) and providing full scholarships for both my MSc (Loughborough, 2007) and PhD (Loughborough, 2017).

To all my teachers throughout my life who have participated in building my teaching and learning skills.

To my supervisor Dr.Lin Guan, who has given me her best guidance and support.

To Loughborough University college of science, department of computer science, and all its student support facilities.

To all my friends, sisters and PhD colleagues in Loughborough.

Publications

Published Conference Proceedings

- AL Amri, S. and Guan, L. (2016). Infrastructure as a service: Exploring network access control challenges. In: 2016 SAI Computing Conference (SAI). [online] IEEE digital library, pp.596-603.
- AL Amri, S. and Guan, L. (2016). Exploring the Firewall Security Consistency in Cloud Computing during Live Migration. In: the 7th International Conference on Computing Communication and Networking Technologies (ICCCNT'16). [online] ACM digital library, article no.40.

To be Submitted Journal

- AL Amri, S. and Guan, L., 2017. "IaaS-Cloud Security Enhancement: An Intelligent Attribute-based Access Control Model". Elsevier Journal of Computers & Security.
  - it has been accepted for the 1st round but then rejected as the information provided is too excessive.
  - the content of this Journal will be divided into two sections. To be submitted to two Journals
- AL Amri, S.M. and Guan, L., 2017. "Empirical Study on the Security of Live Virtual Machine Migration(LVMM) in Cloud Computing". Elsevier Journal of Future Generation Computer Systems.
  - An article answering the question: How an an Intelligent Attribute-based Access Control Model can participate in mitigating LVMM unauthorized access breach in IaaS cloud?

Glossary

ABAC : Attribute-Based Access Control
ACL : Access Control Logic
ACL-DL : Access Control Logic-Deontic Logic
DAC : Discretionary Access Control
DL : Deontic Logic
IaaS : Infrastructure as a Service
JSON: JavaScript Object Notation
LVMM : live virtual machine migration process
MAC : Mandatory Access Control
PaaS : Platform as a Service
PAP: Policy Administration Point
PDP: Policy Decision Point
PEP: Policy Enforcement Point
PIP: Policy Information Point
QoS : Quality of Service
RBAC : Role-Based Access Control
SaaS : Software as a Service
SoD : Separation of Duties
VM: Virtual Machine

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context Background

The cloud computing paradigm is constructed based on several technologies such as virtualisation, Service-Oriented Architecture (SOA), autonomic computing, and grid computing [41]. Cloud computing involves three main services: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). The focus of this research is to enhance an access control model to cope with the IaaS environment without redesigning or modifying IaaS technology structures. Virtualisation technology is one of the main building blocks in deploying IaaS. Hence, the hypervisor affects the efficiency of the access control system. The hypervisor introduces a security threat to the virtual machine (VM) access control because of the single point of access. A trusted VM within an untrusted hypervisor has a higher risk than an untrusted VM within a trusted hypervisor [119]. For example, based on empirical experiments on a live virtual machine migration process (LVMM) [158], it has been recommended to review the existing access control system to avoid unauthorised access during LVMM process. The LVMM process is considered as one of the critical processes in IaaS beside the virtual machine provisioning process [41]. Moreover, access-control security tools in IaaS such as firewalls and security groups cannot support context-aware mechanisms or dynamic access control [8]. The optimisation between the information flow security and IaaS flexibility is getting high consideration as sensitive information maybe leaked to unauthorized entities[227, 203]. Therefore, a well-designed access control system in IaaS is vital.

Security aspects are essential in motivating customers to adopt cloud computing services. For instance, an IDS survey illustrates that 75% of customers consider satisfaction with the degree of security and privacy as the main motivations to adopt cloud computing services [121]. In particular, the access control

aspect is a critical security issue in the IaaS cloud [9, 197]. In contrast to the traditional computing environment, the IaaS cloud has specific characteristics of elasticity, multitenancy, configurability, and dynamicity. Thus, conventional access control models face flexibility challenges and fine-grained limitations when it comes to their implementation and deployment in IaaS [9].

Because IaaS is a multi-tenant environment it is required to handle a variety of user access requirements, accordingly the IaaS access-control system needs to be a context-aware to support fine-grained policy [191]. The traditional access control models such as Discretionary Access Control (DAC) and Mandatory Access Control (MAC) lack scalability and adaptability to dynamic changes. Even though most cloud access control systems such as Amazon, Racspace, Dimansion Data and Verizon use Role-Based Access Control (RBAC) [100], RBAC cannot cope with a dynamic environment because it supports coarse-grained access which creates a restricted access control policy. Also, RBAC access policy rules must be predefined before the access-control process begins.

There is a trend to move from RBAC to Attribute-Based Access Control (ABAC), as the latter is more flexible, supports context-awareness, supports fine-grained policies, and also supports a dynamic computing environment. It is predicted that by 2020, 70% of enterprises will deploy ABAC [70] since there is an interest among industries and governments to deploy access control based on ABAC [99]. Further, ABAC is dynamic and requires less human administration interaction than traditional access control models, because its authorisation process can be computed at the time of the request, where permission is not required to be pre-assigned to users [108]. The findings and analysis of this research match the recommendations of several researchers to use ABAC over RBAC in a cloud computing environment such as in [37, 44, 242].

Figure 1.1 illustrates the scope of this research. The investigation of IaaS security challenges detailed in Sections 2.2 to 2.4 identify a research gap in the area of access control systems. The problem statement will be elaborated in Section 1.2. Four domains of computer science form the basis of this thesis contribution, as will be explained in Section 1.4. These domains are: IaaS cloud, Access Control, Formal Logic, and Artificial Intelligence. Furthermore, Figure 1.1 illustrates the research flow between the four domains.

1.2 Problem Statement

This thesis researches two problems, as illustrated in the tree diagram in Figure 1.2. The first problem is related to fulfilling IaaS access control requirements. Three requirements have been identified. The first requirement is to implement a

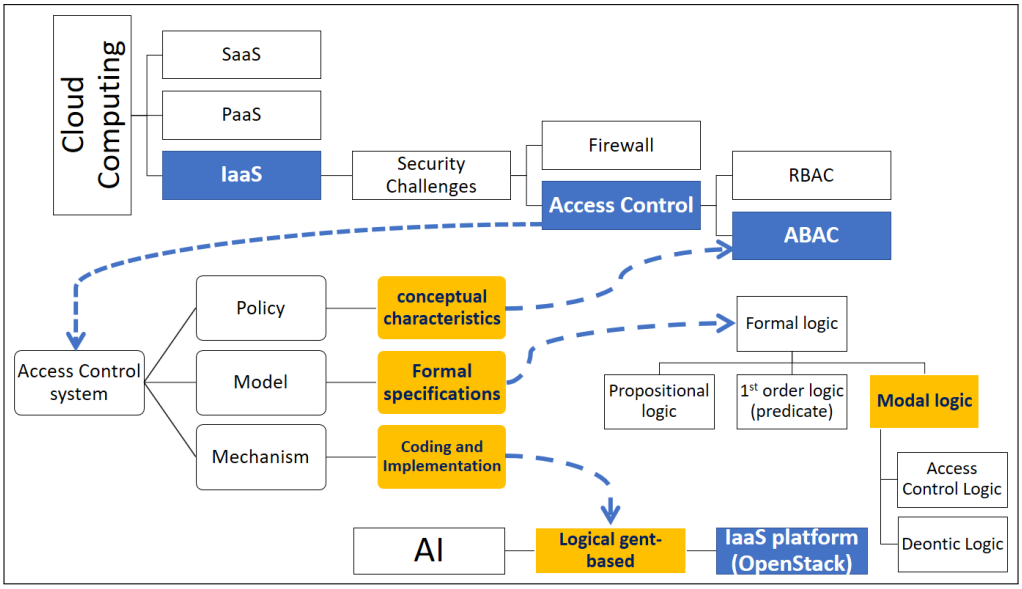

Figure 1.1: Overall thesis research components

context-aware mechanism that supports a fine-grained access feature. The second requirement is to emphasise access-control security principles. With an access control system in cloud computing, the principles of least privilege and separation of duty(SoD) are critical [47, 119, 193]. This research focuses on deploying SoD. The third requirement is to meet the dynamic characteristics of IaaS because it is a changeable environment where users join and leave frequently. Furthermore, the virtual machine configurations change based on the customer's specifications. To the best of our knowledge, these three requirements: context-aware mechanism, SoD security principle, and support of dynamic characters are not addressed in the prior literature on IaaS access control models as will be demonstrated in Chapter 3. ABAC is considered as the most appropriate access control model for IaaS as illustrated in Section 2.5. This thesis works towards solving ABAC challenges discussed in Section 2.6

The second problem discussed in this thesis is related to developing an access control logic that is capable of specifying, verifying and reasoning ABAC models. To the best of available knowledge, a formal logic for ABAC models is not addressed in the prior literature. The required logic should be decidable for it to be implemented in a computing system. The access control logic challenge is two-fold: firstly, it is necessary to find a suitable formal logic category for access control security representations; secondly, a suitable formal language (also called computational language) that can formally interpret ABAC model properties is required. Based on the literature, the modal logic has been found to be the most appropriate formal logic that can present access control models as justified in Section 2.7. Regarding the ABAC, to date, no formal language has been defined.

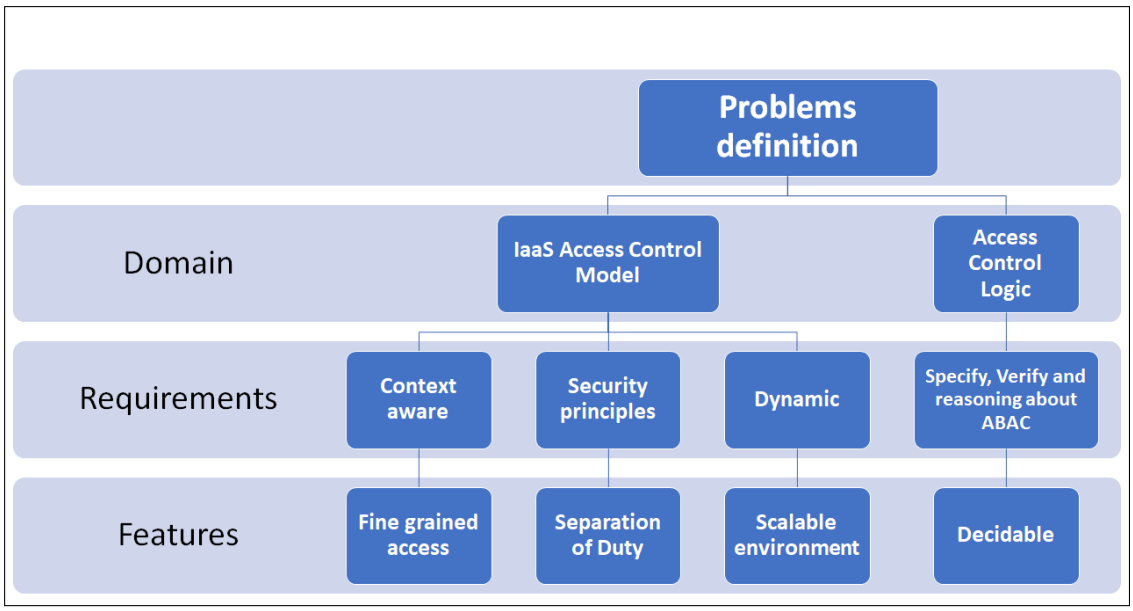

Figure 1.2: Problem Definition of this research

Liner-time Temporal Logic (LTL) and Computation Tree Logic(CTL) are used for model checking. However, LTL formulas are evaluated on paths while CTL formulas are evaluated on states [102]. In order to evaluate attribute based access control system we need to consider states and paths as will be demonstrated in Chapter 5

1.3 Aim and Objectives

Given the challenges that have been identified concerning the access control model in IaaS cloud, this thesis aims to enhance IaaS cloud security by proposing an intelligent attribute based access control model. This goal will be satisfied by completion of the followed objectives:

Obj.1 To identify The IaaS access control model requirements, and explore the limitations of traditional access control enforcement techniques in IaaS clouds.

Obj.2 To investigate how access control models need to change to meet IaaS access control requirements, and identify the access control model best suited to IaaS characteristics.

Obj.3 To propose an access control model enhancement that meets IaaS requirements.

Obj.4 To introduce a formal logic that is able to specify, verify and reason about the proposed access control model

Obj.5 To design an intelligent framework for the proposed access control model that supports the dynamic and scalable features of IaaS.

Obj.6 To implement and test a prototype of the proposed access control model in OpenStack IaaS cloud testbed.

An extensive literature review has been carried out towards fulfilling Obj.1 and Obj.2. As a result, the problem statement in Section 1.2 has been identified. Objectives Obj.3, Obj.4, Obj.5 and Obj.6 are four steps to achieving the main aim of this thesis. The original contributions of this thesis are explained in Section 1.4.

1.4 Original Contributions

Figure 1.3 summarises the contributions of this thesis and the motivation for covering the four computer science fields: IaaS Cloud Security, Access Control, Formal Logic, and Artificial Intelligence. The big picture concerns the enhancement of IaaS security by improving the access control model via the employment of artificial intelligence agent-architecture and modal logic formalisation.

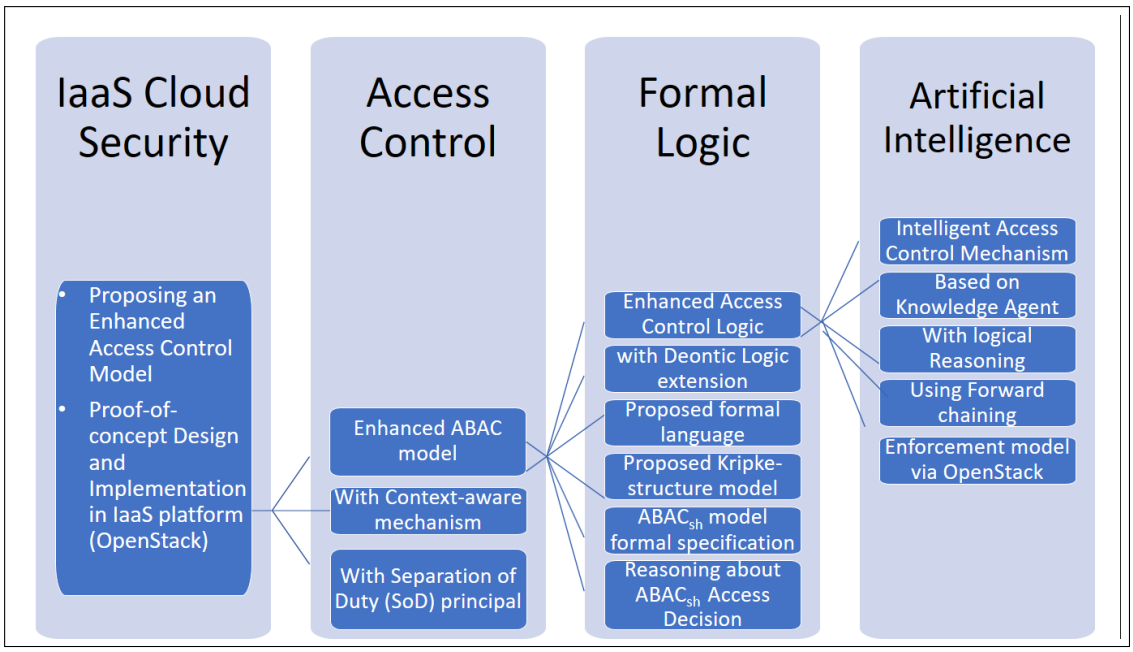

Figure 1.3: Thesis Contributions

The thesis contributions are as follows:

- **IaaS Cloud Security**

  - Propose an enhanced attribute based access control model for IaaS ($ABAC_{sh}$) that supports context-awareness mechanisms and the SoD principle.
  - Propose an enforcement architecture design for $ABAC_{sh}$ based on OpenStack components.
  - Demonstrate $ABAC_{sh}$ implementation in OpenStack. Based on access control QoS metrics, the performance evaluation of `ABAC`$_{sh}$`_PDP` is measured by calculating the time taken to reach an access decision.

- **Access Control**
  - Conduct an extensive study to determine the proper access control model for IaaS.
  - Define the concept behind the context-aware mechanism and the SoD security principle in an $ABAC_{sh}$ model.
  - Perform a comparativeanalysis of $ABAC_{sh}$ with ABAC α and HGABAC that involves $ABAC_{sh}$ computational complexity and QoS (Quality of Service) metric evaluations.

- **Formal Logic**
  - Introduce Access Control Logic-Deontic Logic (ACL-DL) as a modal logic that is capable of specifying, verifying and reasoning ABAC models.
  - ACL-DL combines Access Control Logic (ACL) modality ***says*** with Deontic Logic (DL) modality ***obligation***.
  - Define a formal language and a Kripke-structure model for ACL-DL logic
  - Formally specify $ABAC_{sh}$ via ACL-DL logic.
  - Logically reason $ABAC_{sh}$ access decisions via ACL-DL logic

- **Artificial Intelligence**
  - Propose an Intelligent framework for implementing $ABAC_{sh}$ based on knowledge-agents o
  - The inference engine is based on a forward chaining algorithm.

1.5 Thesis Structure

The thesis is organised into eight chapters. Chapter 2 introduces the literature review, which identifies a need to enhance the access control system in IaaS. The chapter also includes a literature review of concepts used to solve the identified problem. Chapter 3 presents the enhanced Attribute-Based Access Control ($ABAC_{sh}$) and its computational complexity and quality metrics evaluation. Chapter 4 explains the proposed logic that combines the Access Control Logic (ACL) with Deontic Logic (DL). The ACL-DL logic is used to specify, verify and reason about $ABAC_{sh}$, as illustrated in Chapter 5. A proposed intelligent framework for $ABAC_{sh}$ is demonstrated in Chapter 6. A prototype of this framework is implemented and tested on an IaaS platform called OpenStack, as explained in Chapter 7. Chapter 8 concludes this thesis and suggests future improvements and enhancements.

Chapter 2

Literature Review

2.1 Introduction

This chapter investigates and analyses the motivation for this research based on the literature review. It consists of eight sections. As a reference to Figure 1.1 from the introduction, Sections 2.2 to 2.4 explore IaaS security challenges associated with unauthorised access. Section 2.5 studies access control models which leads to a conclusion to build the proposed access control model enhancement based on an ABAC model. ABAC is explored in Section 2.6. The lack of a formal logic that is able to specify, verify and reason about ABAC models leads to Section 2.7, which researches the access control logics. Finally, there is a need to design a dynamic access control which leads to investigating the AI field to propose a framework that is dynamic and scalable. AI in IaaS security is discussed in Section 2.8.

2.2 Security Challenges of IaaS

The characteristics of Infrastructure as a Service (IaaS) encouraged ICT customers to adopt it as the next generation model for outsourcing IT infrastructure [238]. IaaS is capable of controlling and managing the virtualised environment represented in a virtual machine life circle by providing the virtual machine (VM) with the requested resources, such as storage, network and processing power [65].

IaaS features offer business advantages, such as rapid elasticity and fast resource pooling, since IaaS deploys virtualisation technology which supports several innovations, such as multi-core chips and live migrations [41]. A key component in building a virtualisation environment is to operate it via the hypervisor, although the hypervisor on its own cannot build IaaS. Therefore, a cloud-stack is required to build IaaS, such as OpenStack, CloudStack and OpenNebula. According to the current industry, OpenStack is likely to become a dominant cloud-stack [100].

On the other hand, the flexibility characteristics of IaaS introduce several security challenges linked to access control implementation.The elasticity feature of IaaS allows the cloud user (customer) to scale up and scale down to meet their project requirements. This leads to a rapid change in the infrastructure configurations. However, elasticity introduces security challenges in respect to providing an administrative separation between the customer virtual environments. There is a need for a security mechanism that enforces a proper configuration and change management, as well as a fine-grained and predefined access control mechanism [170].

The multitenant nature of IaaS which facilitates the ideal usage of infrastructure by sharing resources between multiple users faces some security challenges as well [100]. Therefore, it is more likely that there is a need for a new type of access control policy between tenants in intra-cloud communication [173]. A tenant can be an enterprise in the context of a public cloud or a department within an enterprise in the context of a private cloud [26].

The flexibility feature of IaaS enables the user to configure their own virtual machines and computing infrastructure [237]. Hence, it is prone to misconfiguration that can lead to a security violation [27]. Therefore, there is a need to monitor cloud behaviour to figure out unexpected errors. For example, in April 2011 an infrastructure outage caused Amazon's Compute Cloud EC2 to be unavailable for its customers [63]. Therefore, there is a need to monitor IaaS behaviour. In the literature, an approach constructed on role-based access control has been proposed [177].

The dynamicity of IaaS facilitates virtual machine (VM) mobility among physical machines for different aspects, such as server consolidation, load balancing, data recovery and green computing is achieved through a technique called live virtual machine migration (LVMM). LVMM allows the movement of virtual machines between the physical machines during run time with a minimum downtime [67]. Although live migration supports IaaS dynamicity, it introduces some security threats. The protocol used for live migration moves the virtual machine state in plain text, which allows hackers to snoop it through the network links. Even encryption is not able to secure it, as illustrated experimentally in [158]. Therefore, there is a need for re-thinking the existing access control and isolation mechanism.

Moreover, the dynamicity feature of IaaS affects firewall functionality since the VM gets a dynamic IP address through a DHCP server that assigns a lease time for the client IP address. The virtual machine must renew its IP address when the lease time expires. Upon renewing its IP address, it may or may not receive the same IP address that it received previously [128]. A source IP address and

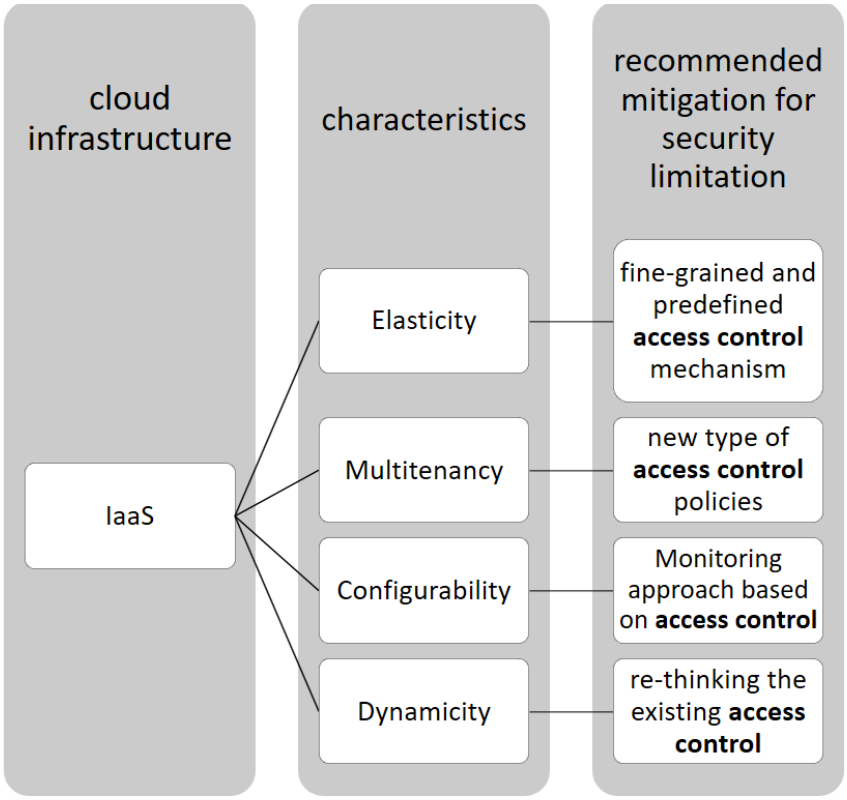

Figure 2.1: Recommended mitigation for some of IaaS security challenges

a destination IP address are basic information to generate a firewall rule. The firewall rules remain constant unless there is an explicit need to change the policy, as a result the firewall cannot adapt to real-time threats [210]. Any system that uses predetermined and fixed IPs might impose some limitation on the dynamism and the scalability of IaaS [215].

A firewall is a critical network access control mechanism, therefore it will be discussed in more detail in Section 2.4. The above IaaS security challenges analysis shows that there are several security vulnerabilities occurring as a side effect of the substantial IaaS characteristics. Figure 2.1 reveals that a proper design and implementation of access control is a critical element that can contribute to mitigating the majority of the security IaaS challenges.

2.3 Example of a Security Breach in IaaS During LVMM Process

The IaaS layer consists of several components, such as virtualization, networking, storage and processing. Virtualisation is one of the key components; furthermore, the main core services in IaaS are Virtual Machine (VM) provisioning and VM migration [41]. Security of the live virtual machine migration (LVMM) is considered

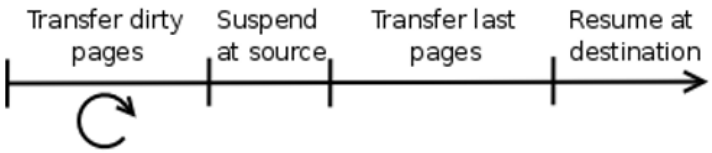

Figure 2.2: LVMM Process Steps [204]

one of the major challenges in IaaS and a critical research topic [13, 41, 148, 222].

There is a default LVMM algorithm in most popular hypervisors, such as Xen, VMWare and KVM [141]. A typical LVMM mechanism consists of four main stages, as shown in Figure 2.2 , which start with several iterations that aim to transfer VM memory pages from the source physical machine to the destination, where a new location for VM has been selected.

The ideal migration process is to copy the complete state of the VM, including memory, disk and network connection [141]. From a networking point of view, there are two categories of LVMM: moving VMs belonging to the same sub-network and moving VMs between different sub-networks. The latter requires a change of IP address and introduces another challenge that limits migration in cloud computing [208].

LVMM has been penetrated via man-in-the-middle attacks, as well as successfully attacked by through flag migration [158, 218, 66]. LVMM introduces important security issues because it includes VM state transfer through communication links, which can be attacked by ARP spoofing, DNS spoofing and route Hijacking [172].

Live Virtual Machine Migration (LVMM) security vulnerabilities are related to the security exposures in virtualisation technology [13, 91, 172, 190, 223, 235, 241]. The following illustrates LVMM vulnerabilities, which have been categorised into three sources of threat, as follows:

- The first vulnerability is through the network link, as live migration involves a lot of network state transfer. Encryption techniques can be involved in mitigating this risk, but there should be a careful consideration of downtime since LVMM runs in real time. IPsec has been used in some security approaches to mitigate live migration threats, although it introduces a computational delay [201, 194]. Moreover, encryption is not able completely to mitigate this security hole [158].

- The second vulnerability is through the host (physical machine), where it can be attacked or it can host an untrustworthy VM. As a consequence, the migrated VM security can be compromised since it will be in a shared environment with malicious components. Attacks can also be initiated in

migrated VMs through the hypervisor if its corresponding host has been successfully compromised.

- The third vulnerability is based on security configuration consistency and efficiency. This is as a result of the different natures of physical appliances and virtual appliances, such as a firewall. This vulnerability occurs due to various factors, such as firewall placement and the policy configurations [228]. Moreover, elasticity introduces security flaws caused by misconfiguration after a migration is triggered [105]. Network state consistency can also be affected after migration where some network packets might be lost during LVMM downtime [130].

To sum up, LVMM main security issues arise as a consequence of an environment shared between different customers (cloud users), which is a multi-tenancy feature of IaaS, as well as dynamicity and elasticity features which raise the need to update access control policies as the virtual machine changes its location. We can conclude that LVMM security is a critical issue in IaaS and it faces serious security challenges that need to be addressed and considered as an open research topic.

There is a trend to secure LVMM through vTPM [58]. Nevertheless, TPM-based measurements are ineffective for detecting a malicious cloud service provider as well as having limitations in verifying the hypervisor integrity in public clouds via remote attestation [33]. To secure LVMM, there is a need to design an access control policy that allows the administrator to manage migration privileges. The existing access control model in IaaS should be upgraded to cope with the emerging security challenges [158]. Similarly, existing firewall approaches should be modified to meet IaaS characteristics.

2.4 IaaS Firewall System

IaaS dynamicity and rapid infrastructure changes, due to adding and removing virtual machines and virtual machine migration, introduce a challenge to the firewall as there is a need to update firewall entries frequently. This leads to increasing maintenance overheads as firewall policies need to be updated in such large scale environment [173]. If the firewall is not well constructed, managed and updated, the IaaS will be at risk, resulting in facilitating for hackers the access to the cloud interface on behalf of legitimate users [33, 60].

2.4.1 Security Group Approach

A firewall system is a critical network access control enforcement mechanism in most computing environments. A cloud service provider (CSP) provides firewall functionality in the form of a security group. For example, Amazon, Windows Azure and OpenStack implement the concept of the security group to provide a firewalling service to their customers.

In general, security groups are set to deny everything by default and individual services must be enabled by the client. The security group allows customers to restrict traffic to and from their VMs. All VMs which belong to the same security group will have the same firewall policy [100]. This makes a cloud firewall service relatively user-friendly, but it lacks many of the features commonly found on local firewall products [54]. The security groups alone are insufficient to prevent attackers from communicating with the external network [126].

There are two methods of setting up firewall policy through security group: either to create an entry for each VM in the security group or to group VMs in one entry based on their IP prefix. The first method faces scalability limitations in IaaS, while the second method complicates VM address management [173]. Therefore, cloud can bring some potential security threats to the organisation by not having an organisation specific firewall [132].

2.4.2 IaaS Firewall Security Threats and Limitations

As IaaS provides several features that introduce flexibility into the cloud infrastructure, it initiates complexity in firewall configuration and installation. Due to the large scale of IaaS infrastructure, a simple policy can lead to a large number of fine-grained rules [147].

The effectiveness of firewall security depends on its policies. However, firewall policies are often error prone due to the complex nature of firewall configurations as well as the lack of systematic analysis mechanisms and tools [95]. In a distributed environment, detecting anomalies in firewalls has become a complex task [79]. According to an empirical study of middle-box failures over two years in a service provider network, it was found that firewalls crash more than other security systems, such as IPDS and VPN, and that 33% of firewall failures are due to misconfiguration, faulty failovers and software version mismatch [175]. Therefore, the firewall policy management area is an evolving research field, as policy correctness and consistency among firewall systems is an essential element for enhancing firewall security [64, 184]. To get a cloud firewall policy configuration pattern, intensive experiments are needed to make security policy complete [124].Therefore, several researchers are concerned with improving firewall policy

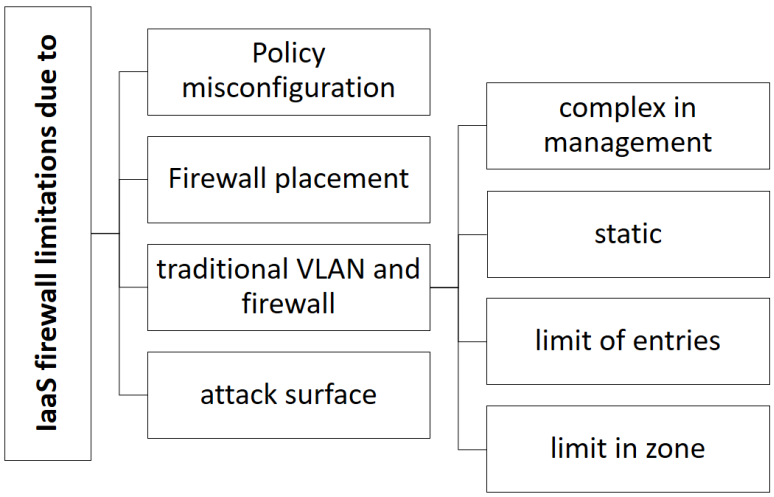

Figure 2.3: IaaS firewall limitations

strategy; for example, the Tree-Rule firewall, which uses the NF-IP-FORWARD algorithm to improve the performance of firewalls in cloud policy configurations [92].

Another problem with firewalls in cloud computing is their placement; too few firewalls can cause a large number of communication flows. The placement of firewalls in cloud computing is critical in maximising the security benefits they offer. Traditional firewall placement is not sufficient in cloud computing, as it will introduce traffic overhead to the network switches and hypervisors [147].A major network security risk in cloud computing is due to the limits of traditional firewall connections [71]. Traditional firewall settings are not sufficient for optimal fine-grained decisions and application-level as they are not able to deal with dynamically opened server ports for encrypted connections [226].

Several researchers consider firewalls and VLAN to be less effective in the cloud environment, due to several challenges such as time consuming in configuration and management, limitation of the geographic zones, limitation to the number of users and static nature [130, 155, 173, 215].The firewalls can be breached in cloud environments by a mechanism using UDP coordinating with TCP [133]. An analytical experiment shows that there is a time interval where LVMM is not under firewall protection. A firewall cannot differentiate normal traffic from attack traffic if it accesses the network through port 80 [146]. Moreover, 42% of firewall failures are due to DDoS attack at the network layer [175]. Traditional packet-level firewall mechanisms are not suitable for cloud platforms in cases of complex attacks [232].

The limitations of firewall configuration in cloud computing according to this investigation are summarised in Figure 2.3. We can notice that traditional access control enforcement mechanisms are not sufficient in cloud computing. Therefore,

there is a need to redesign a firewall system that suits IaaS characteristics.

2.4.3 Firewall Approaches in IaaS

Cloud users, such as companies and governments, might not rely on cloud-based firewalling approaches as these approaches still experience severe performance and reliability issues [87].Ensuring network security in the current complex infrastructure which involves different vendors and cloud service providers (CSP) turns out to be difficult and time-consuming. Each CSP has their own API to manage the security mechanisms, such as traditional firewalls, virtual firewalls and security tools. Therefore, what looks like a simple application change may require tens or even hundreds of configurations [90].

As shown by Table 2.1 , IaaS firewall approaches from the literature agree that the traditional firewall needs to be improved or replaced in order to cope with the IaaS environment. A trend of deploying virtual firewall is illustrated in most of the approaches. Firewall virtualisation allows dynamic deployment, so it suits IaaS characteristics and it can effectively improve firewall configuration [87, 224].

Most of the academic research assumes that the insiders in cloud service providers are not trusted [100, 132, 210]. To improve the trust in the cloud environment, a bridge virtual firewall can be designed and installed on the virtual machine in IaaS so that the cloud user can have full control on their firewall [132]. A bridge firewall improves the performance, but it limits the security of the live migration as this type of firewall cannot manage different types of policies. Moreover, massive attacks may compromise a virtual firewall if they originate from outside the virtual domain [87].The virtual firewall side effect can be mitigated if a proper firewall is designed among virtual machines and suitable firewall policies are defined [124].

Table 2.1 indicates the two approaches discussed regarding firewall systems administration in IaaS, which are centralised and distributed. The centralised approach was found to be less prone to misconfiguration failure as it monitored continually [175]. On the other hand, it has several drawbacks: it may lead the centralised controller to reach a bottleneck, it attracts more DDoS attacks and can introduce a single point of failure [173]. The centralised firewall set-up is argued to be unfeasible in the cloud due to performance and cost issues [129]. The distributed approach is used by many enterprises on the network edge [147]. To handle dynamic policy update in IaaS, a distributed firewall needs a complicated revocation and re-propagation mechanism [96].

Table 2.1: Firewall approaches in dynamic network

Ref	Main Problem	Proposed Solution	Admin Type
[173]	Conventional network access contorl: firewall and VLAN face several limitation in IaaS	Take the policy enforcement point out from the network and place it into the hypervisor	Centralised
[96]	Network states and traffic are frequently changed	Firewall Framework to cope up with SDN envirnomnt	Centralised
[147]	fine-grained rules are needed by CSP to get better control over individual network flows	For better scalability and performance, place the access control policy on both hypervisor and switch	Centralised
[87]	Cloud firewalling suffers from performance and reliability issues	Propose a framework consisting of phyical firewall and virtual firewall	Centralised
[232]	Packet level firewall mechanisism can not handle a complex attack on the cloud	Cloud firewall framework involves event level detection chain with dynamic resource allocation	Not mentioned
[210]	Network topology is not well defined, insiders are not trusted. Policies in Conventional firewalls are static and can not adapt to real-time threats	Propose a distributed firewall with a distributed active response by moving policy enforcement point from network firewall to end host	Distributed
[132]	CSP is not fully trusted	Propose a firewall system monitored by the cloud customers	Distributed

2.5 Access Control Model

Authentication, access control, and audit together provide the foundation for information and system security. Consequently, access control is applied after authentication has been established [189]. In cloud computing, the authentication technique is fulfilled through identity management that supports access control based on user attributes [212]. Vaquero studied several virtualised (multitenant) datacentres and concluded that most reported systems employed access control techniques to secure their environment [215].

As has been discussed in IaaS security challenges in Section 2.2, it was found that an appropriate access control mechanism is needed to mitigate most of the explored threats. Unfortunately, the classical access control models such as mandatory, discretionary and role-based are not suitable for IaaS due to its characteristics [19, 110, 145]. Several attributes should be taken into consideration to set up proper access control for the cloud environment [103, 137, 173]. These are:

- The method of access to the cloud and cloud architecture. The users in the cloud are identified by their attributes or their characteristics, not by fixed IP address. Therefore, a dynamic access control is needed to achieve cross-domain authentication. The cloud access control should be network independence.

- The multi-tenancy feature in IaaS requires flexibility in seating access policy as different users are sharing the same infrastructure, although they are most probably not from the same organisation or country.

2.5.1 Background and Challenges

As a general principle, an access-control model aims to protect objects from unauthorised subjects based on a specific access control policy. Authorisation means that an authenticated subject is allowed to perform an operation on an object. Each access-control policy has a model that it follows. Each access-control model consists of a well-defined framework and a set of boundaries that describes the relations used to combine the objects, the subjects, the operations and access rules in order to generate an access-control decision. There are four main access control models: Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC). Several research scholars have described these access control model's properties and their limitations [10, 20, 94, 109, 113, 125, 137, 138, 189, 233, 236]. Access-control models can be categorised based on the main entity that is involved in the creation of the access policy rules. For example, the entities can

be the subject or the object. Identity-Based Access Control (IBAC) categories, such as DAC and RBAC, face scalability challenges in a large system since they are based on the subject identity. Moreover, they are cumbersome and lack the ability to adopt dynamic policy changes. DAC has two main limitations: the first limitation relates to delegation issues since its read access is granted transitively. The second limitation is due to its access structure concept being dependant on subject identification. Therefore, it allows the program to inherit the identity of the invoking user, which is considered a security vulnerability. RBAC has been standardised and used widely in most economical computing systems [76]. Its role is considered as a link between users and permissions in order to permit or deny access [47]. RBAC faces several limitations, such as role expansion in dynamic systems. It is static as its access rules must be pre-defined before the system is running [59, 94]. Lattice-based access control categories, such as Mandatory Access Control (MAC), are centralised models where the security administrator assigns fixed security labels to objects. MAC faces similar scalability challenges to RBAC and DAC. The last access-control model category is based on system attributes such as Attribute-based Access Control (ABAC). It controls access by evaluating rules against the attributes of several elements, including the entities (subject and object), the operations, and the environment. It is the most suitable candidate for the IaaS access control model because ABAC is scalable and supports a dynamic environment.

The challenges facing the design and implementation of the access-control system are as follows [21, 89, 94]:

- It is a complex system that is divided into three layers: policy, model, and mechanism. The interaction and the relation between these is sophisticated
- Translating the business policy to the access control rule set is a complex process. It requires an expressive tool such as formal logic that is able to specify the policy statements into rules that can be computed by machines
- There is no perfect access-control model that will fit all types of computing systems. Each access control model involves a trade-off and limitations. Therefore, the selection of the proper model is based on the system requirements and the business functions
- The access control state is safe if no permission can be taken by an unauthorised principal. The safety computation has been proven to encounter an undecidable problem in traditional access control models such as access matrix.

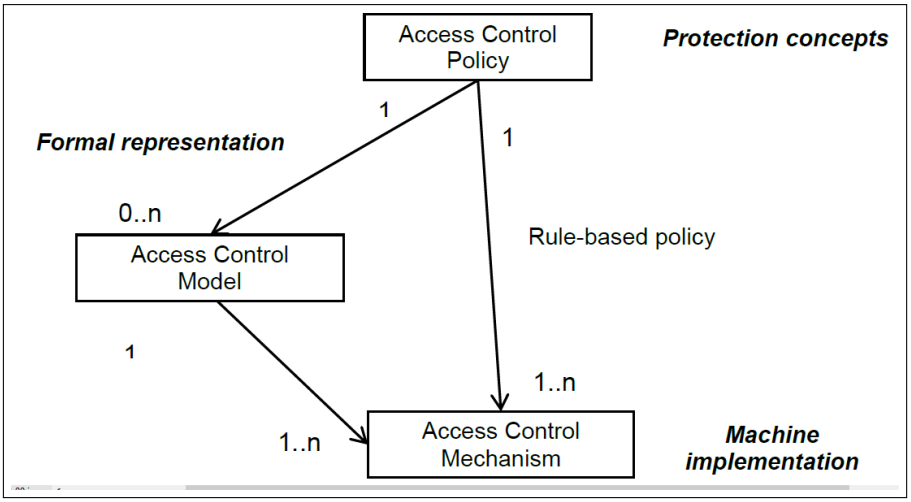

Figure 2.4: Access Control building blocks as explained by [97]

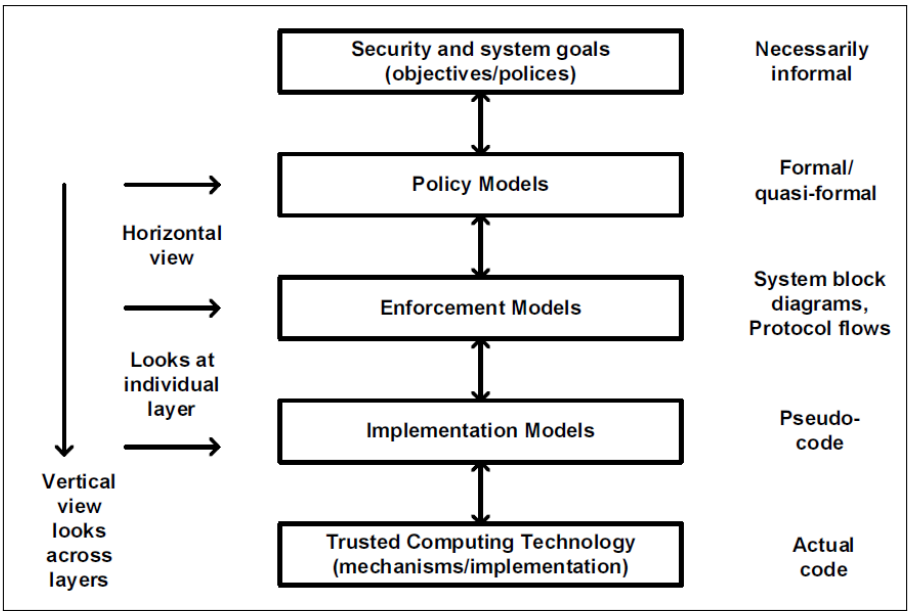

Figure 2.5: Access Control building blocks as explained by [187]

In order to facilitate the process of designing and implementing the access control system, it has been divided into three layers: policy, model, and mechanism. The definitions of these three building blocks are explained below [10, 21, 94, 98, 104, 186, 187].

Access Control Policy.To distinguish the access-control policy from other usages of the term 'policy', it is defined as a specification of the system behaviour [16]. This will produce a set of Allow/Deny rules that reflect the business owner's objectives and security policies.

Access Control Model. This is an intermediate layer between the higher level, which is the policy, and the lower level, which is the mechanism. In some cases, the policy is implemented directly into the mechanism without formally

defining the access-control model if the policy is application-specific and implemented by a particular vendor. However, in cases where the access-control policy spans multiple computing platforms or consists of several security factors, there is a necessity to formally define an access-control model. Mainly, the model specifies how the policy can be implemented and enforced via the access-control mechanism.

Access Control Mechanism. This involves functions that implement the control obligated by the access-control policy stated in the model. It can be written in the form of pseudocode which can be implemented in a hardware layer or a software layer by using a programming language.

Figure 2.4 illustrates the access control building blocks by [97]. The policy reflects the protection concepts where the model can formally be specified. However, Figure 2.5 bring the term policy model which can be formally specified by [187]. In [206], Tang used the term Policy to refer to his proposed ABAC enhancement for multi-tenant IaaS where the term Enforcement model was used to refer to his proposed framework. It was found that the term 'access-control policy'is used interchangeably with the term 'access-control model'. However, it is common to refer to RBAC, ABAC, DAC and MAC as access control models instead of access control policies as Figure 2.6 shows results from Google scholar.

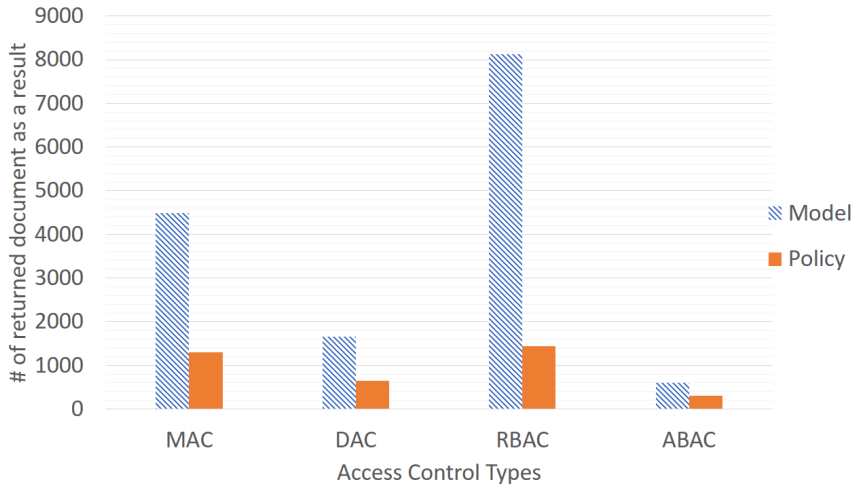

Figure 2.6: Google Scholar Results for the keywords Model vs. Policy for Access Control Types

It can be concluded that the term 'policy'is used to describe the architecture that is followed to create access rules, where the term 'model'refers to the formal specifications that are used to express the access control system properties. The access-control models: MAC, DAC, RBAC and ABAC are occasionally referred to as policies since they indicate how the access rules are created. MAC rules are based on object-labelling. DAC rules are based on subject-identity. RBAC rules are based on subject-roles. ABAC rules are based on attributes. This research

will refer to them as 'access control models'to avoid ambiguity, and will use the term 'policy'when discussing authorisation rules.

2.5.2 Access Control Models Approaches in IaaS

Several access control models are proposed in the literature which address various access control problems in IaaS. Table 2.2 summarises IaaS cloud access-control models which have been published recently. The most common limitation found among them is the performance overhead, which leads to an increase in computational cost that affects QoS (Quality of Service). The access control policies and models presented by Nguyen and Tang [152, 206] use RBAC and ABAC as a basis for their proposed models. Their architectures are implemented via XACML and provide a proof of concept via OpenStack.

Table 2.2: Recent proposed Access Control Models for IaaS

Ref.	Access Control requirement	Proposed solution	Used tools
[152]	expressive and fine-grained access control as prior models are not suitable	PBAC (Provenance-based Access Control) a sub-set of ABAC	XACML and OpenStack
[206]	a fine-grained access control model for multi-tenant collaboration	Multi-Tenant Role-Based Access Control (MT-RBAC)	XACML and OpenStack
[1]	support multi-tenant hosting and heterogeneity of security policies, rules and domains	Access Control model for Cloud Computing (AC3)	theoretical
[14]	manage access from users or entity that belong to different authorization domains	Capability-based Access Control	theoretical
[122]	flexible and fine-grained access control for VM-level	IaaS-oriented Hybrid Access Control (iHAC)	iVIC as IaaS

continued on next page

Ref.	Access Control requirement	Proposed solution	Used tools
[123]	a user-centric access control for mobile Cloud environments	Context-Aware Attribute-Based Techniques	XACML
[239]	balance between information security and resource sharing	hierarchical secure information and resource sharing model	OpenStack
[27]	virtual resources in IaaS are configured using software, and hence prone to misconfigurations	attribute-based constraints specification and enforcement	OpenStack
[150]	Multi-tenant access control is not supported	ABAC that support multi-tenant access	OpenStack
[134]	centralized access mediation and flexible policy customization	least-invasive access control framework	OpenStack
[22]	The relation between users and resources in cloud computing is ad hoc and dynamic	Extend RBAC with trust relation	CloudSim in Planet Lab

The proposed model by A. Younis *et al* and Anggorojati *et al* [1, 14] presents a great work regarding security policy design based on attributes, but it does not present a security model architecture. The proposed model and the enforcement implementation presented by Li *et al* [122] is remarkable, as it does not consider IP address as the main attribute in the access-control policy, instead it uses security type.

However, their model does incur a performance overhead. A concentration on multi-tenant access-control is investigated by Ngo *et al* [150] and an ABAC model is proposed to cope with this feature. A cloud user-centric access control model is proposed by Li [123] which targets customer privacy within mobile cloud applications rather than IaaS cloud. Luo *et al* in [134] investigate the improvement of access control models in an OpenStack IaaS cloud. RBAC is still used in most cloud computing, Behera and Khilarin in [22] extend it with a trust-relation.

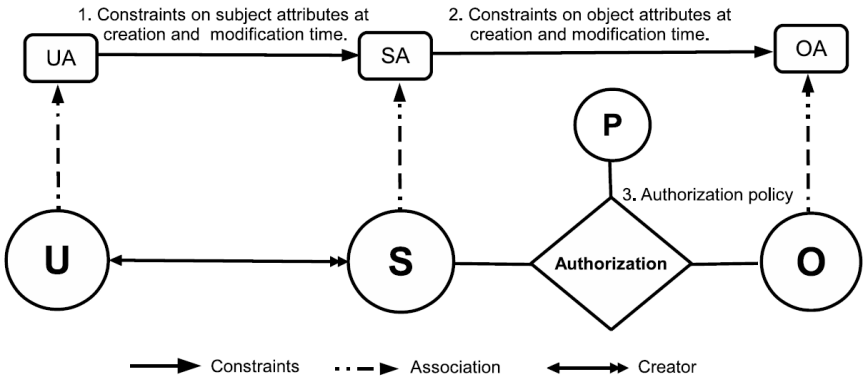

Figure 2.7: Unified ABAC Model Structure [108]

2.6 Attribute-Based Access Control (ABAC)

The definition of Attribute-Based Access Control (ABAC) in this section is extrapolated based on the knowledge obtained from various ABAC references such as [52, 98, 221].

Attribute is the base element in calculating the access decision in ABAC models as illistrated in Figure 2.7. UA is the user attributes, SA is the subject attributes and OA is the object attributes. Attribute is defined as a pair of (name: value), where value can be a complex data structure. They can be associated with different entities such as users, actions, subjects, objects, context and environment. Policy rules convert the combination of attributes into access rights in real-time. Therefore, the permissions in ABAC are not listed beforehand. The use of predefined attributes instead of predefined permissions in decision making adds flexibility to the access-control system. For example, in RBAC, the permission engineering process requires the grouping of permissions into roles; users are then assigned to these roles. However, as the system scales up, these processes become more complex and face a roles-explosions problem [98].

One of the critical differences between ABAC and previous access control models is that ABAC access decisions are not associated with user-identity, making ABAC dynamic and able to cope with changes. Therefore, if the system administrator intends to update access control policy, the only place which must be changed is the policy-rules file. In contrast, it is difficult to determine which places need to be updated in identity-bounded models such as ACL (Access Control List) which is used with DAC and RBAC models.

This thesis defines ABAC as follows: ABAC generates its access control decision based on a collection of attributes driven from the involved entities in the access-request and the system environment. Attributes enable the ABAC model to be fine-grained, flexible and able to express a complex Boolean rule-set. ABAC

supports a real-time access-decisions and a dynamic access control system since it is identity-free based.

2.6.1 Background and Challenges

ABACα is described in the literature as a foundation of the ABAC model's functionality. Therefore, it is essential to identify ABACα limitations so as to be able to propose enhancements to ABAC. The list of challenges is stated below [108, 118, 192, 193]:

- ABACα focuses on the ABAC basic with minimal features. It does not take into consideration the object and the subject hierarchies
- It does not involve environment and context attributes
- It does not formally express the fundamental access-control security principles such as separation of duty (SoD)
- It does not take into consideration the delegation feature
- It is considered as one of the first contributions in formalising the ABAC model, however, the policy language used is insufficient for real-world applications such as handling multiple policy compositions or policy conflict
- The authors in [195] present ABACα in OWL. Their investigation concludes that ABACα does not cover static or dynamic separation of duty principles nor does the subject involve an attribute that reflects its contextual information

More precisely, the ABACα author recommendations are as follows:

- The proposed Common Policy Language (CPL) is not a complete language as it does not specify the values of the symbols set and atomic variable. These two values have a significant effect on each configuration point. Therefore; there is a need to investigate a policy language that is able to detect misconfiguration and can present compliance with privacy expectations
- Attribute engineering might add complexity to ABAC. In addition to the difficulty in policy expression and comprehension due to the policy configuration point proliferation

It can be deduced that ABAC challenges are two-fold: The first is related to its conceptual definition and the second is related to its formal definition. These

two challenges are detected as well by [118, 188, 193]. The conceptual definition includes the features supported by the ABAC model, such as the separation of duty security principle and the context aware mechanism. The formal definition includes a formal language that is able to formally specify ABAC model properties.

The use of formal logic in the design of ABAC policy rules is a convenient approach for two distinct reasons [30]. The first being that formal logic is straightforward since it does not have an up-front cost, unlike RBAC role engineering. The second reason is that formal logic is flexible, since its attributes facilitate the expression of complex policy rules.One of the earliest works related to the logical based framework for ABAC was carried out by Wang *et al* [221], using the constraint logic programming language. Bijon *et al* in [28] developed a declarative language for Attribute Based Constraint specification (ABCL). Hence, the declarative language is specific since it focuses on specified constraint values of the attributes held by the ABAC model. However, ABCL did not specify the values of the attribute related to authorisation policy rules.

There are several policy specification languages, including SecPAL, DYNPAL, SMP, Binder, SPKI/SDSI, XACML and EPAL. However, they all focus on specifying and evaluating authorisation policy, and none present comprehensive formal models for ABAC. Jin in [108] defined a template called Common Policy Language (CPL), although it is not a complete language as the symbol set and atom are left unspecified. There is an open research in policy specification languages related to misconfiguration detection and privacy compliance [108]. Most of the logic-based formalisations presented for ABAC lack the ability to manage attributes and engineer security policy rules [118]. Considerable attention is given to improving the use of XACML as a policy language for ABAC because it is compatible with many web applications. However, XACML does face limitations related to policy semantics [195, 200]. An Interesting work was carried out by Crampton & Williams in [53] regarding the completeness of the ABAC model using XACML formal logic, where they introduce the use of three-valued policy in XACML.

The techniques used for the ABAC policy specification are either formal logic or enumeration. Examples of the first category are ABACα, HGABAC and XACML. Examples of the second category are Policy Machine (PM) and 2-sorted-RBAC [30]. This study selects the formal logic technique for ABAC specification since it allows the research to verify and reason ABAC. The Section 2.7 justifies the selection of formal logic approch.

2.6.2 ABAC in IaaS Cloud

The ABAC model concept is context-aware in the sense that it can utilise the attributes concept in its authorisation decision. However, ABACα [108] does not indicate the context-aware feature. Instead, the author contains some concepts of context in his ABACβ which is designed for the purpose of unifying RBAC extended models. Smari *et al* [198] extended ABACα with context, privacy, and trust features where context is involved in access-control policy beside subject and object. Thus far, there is no illustration regarding system complexity evaluation nor policy language formalisation. Servos and Osborn [192] proposes HGABAC where the ABAC model takes into consideration hierarchal features. Servos & Osborn used some context-awareness by involving the network-connection and environment attributes. However, they did not consider the separation of duty concept or delegation feature. Their policy language is also based on Kleene K3 logic. The formal languages used in ABACα and HGABAC are a type of propositional logic. Therefore, they face the satisfiability problem of NP-complete during policy update or policy review. Even if a formal language based on first-order logic is used to express the ABAC policy, it will face an undecidable computational problem [30].The focus of this research is to specify the context-aware mechanism and separation of duty security principle in ABAC model.

2.7 Access Control Model Formal Specification, Verification and Reasoning

2.7.1 Formal Logic in Access Control

Access control is an essential and widely used security mechanism. One of the critical challenges in implementing access control is its policy specification and management. There is a limitation in the available tools that allow administrators to mitigate this challenge. Therefore, policy misconfiguration is the cause of a huge number of security breaches cases [106].

Based on the literature, access control specification can follow a graph-based approach or logic-based method. The graph-based approach is used to represent access control models. However, the logic-based method is more convenient when it comes to reasoning and computing [24]. Logic-based access control models are suited to large dynamic systems as they are expressive and able to represent a variety of authorisation policies [93]. Using formal logic for specifying, reasoning and verifying access control systems is an active research area. The logical approach is more appropriate for this access control research scope than calculus-

based approaches since it allows the detection of patterns of correct or incorrect reasoning [16, 47].

In early 1973, the status of logic in computer science was not clear, and its usefulness was not identified because of a lack of mechanical testing to validate predicate logic. Furthermore, Boolean logic is intractable. Classical logic was established to provide a formal language to describe how mathematics works using proven methods [45]. Classical logic (also known as sentential logic), such as propositional logic and predicate logic (also known as First Order Logic or FOL), formulas are either true or false in any model. Non-classical logic such as modal logic (also known as symbolic logic) differ from classical logic as they replace the concept of truth, which consists of two values: true or false, with the concept of constructive provability. The constructive provability operations preserve justifications with respect to evidence and provability rather than truth value.

In natural language, there are various truth modes such as it is believed to be true (doxastic logic), it is always held (temporal logic), it is known to be true (epistemic logic), and it ought to be the case that it is true (deontic logic) [102]. Therefore, modal logic is more appropriate to reason in a dynamic situation where a truth value varies over time. On the other hand, classical logic has a static neuter [219]. First order logic might be more expressive then modal logic, but in term of validities, modal logic is decidable where first order logic is undecidable. Therefore, modal logic strikes a balance between expressive power and computational complexity [214]. Table 2.3 compare modal logic with classical logic. [214] Modal logics tend to be decidable ; and their validities can be described in transparent variable-free notations. Understanding the trade-off between the expressive power and axiomatizability is essential in formal logic reasoning.

Table 2.3: Comparing Modal Logic with Classical Logic

	Classical Logic	Modal Logic
Value nature	Static	Dynamic
Value Range	Either False or True	Concept of constructive provability
Based on	Fixed Truth value	Truth value varies over time
Expressiveness	More	Less
Computation complexity	Less	More

2.7.2 Decontic Logic (DL)

Modal logic extends the classical logic operators with modal operators that allow the representation of possibility and necessity [68]. Deontic logic is a type of modal logic that involves operators such as obligations, prohibitions and permissions. It is used in normative systems such as computer security, legal systems and electronic commerce [39]. Information Systems (IS) store real-world data. The property that exists in this world that is known to be true is called an integrity constraint. Different logics can be used for normal (hard) constraints. On the other hand, the exceptional (soft) constraints which are the desirable properties can be violated. Therefore, there is a demand for a logic that is able to capture soft constraints such as deontic logic, because deontic logic has an advantage of reasoning between the ideal and the actual behaviour [49]. Furthermore, deontic logic supports formal languages in avoiding the ambiguity of access control policy specifications and is consistent because its soundness and completeness are achieved through its axiomatic characteristics. As a consequence, it supports an access control implementation level [144].

Deontic logic receives a large degree of attention in the field of computer science [6, 181]. Chevy & Cuppens [49] presents one of the earliest logical reasoning approaches to the properties of security policy in access control based on deontic logic. Furthermore, Glasgow *et al* in [85] propose a formal framework for specifying and reasoning about security policies called Security Logic (SL), where deontic logic had been involved in the formal language used. However, SL did not involve any access control logic operators. An application of deontic logic in Role-Based Access Control (RBAC) is introduced by Koaczek [116]. Another work using deontic logic in access control was presented by Abou El Kalam [5], where a framework is proposed that can help the system administrator to automatically derive the consequences of their policies.

2.7.3 Access Control Logic (ACL)

Several researchers contribute in defining and establishing Access Control Logic (ACL). Chin and Older [47] use similar semantics to the logic of Abadi and his colleagues, who have several contributions in this field such as [4, 2, 3, 120] and their cooperation with Deepak [82, 81]. The ACL conducted by Abadi is considered as one of the foundations for ACL researchers. However, ACL has not been used to formally define ABAC in the literature. Boella *et al* [34] proposes a logical framework named Fibred Security Language (FSL). FSL aims to study ***says*** operators. The FSL is based on a multi-modal logic methodology which uses a fibring mechanism to combine logic based on the work of Gabbay [80].

Several researchers have used intuitionistic logic to enhance ACL. Intuitionistic logic is also known as constructive logic. Genovese *et al* [84] propose an ACL modal (M-ACL) based on intuitionistic multimodal logic. M-ACL extends intuitionistic propositional logic with ***says*** binary modality and ***speaks-for*** binary operator between principles. Genovese *et al* prove the language soundness and completeness using first order logic. Genovese *et al* in [83] present another modal called ACL+ for access control policy specification, reasoning, and enforcement to overcome specific limitations of the basic ACL. These limitations are implicit permissions, control or delegable permissions, and information flow versus acceptance. Massacci [139] presents a logic for practical reasoning with RBAC based on Abadi ACL in addition to a decision method based on analytic tableaux. Whereas Kosiyatrakul [117] extends Abadi ACL, with four relations to reason about an RBAC policy model.

2.8 Artificial Intelligence in IaaS Cloud Security

To the best knowledge of the author, no prior work addresses the use of a knowledge-agent in access control model implementation in IaaS. The knowledge-agent is a type of logic-based agent; therefore, it suits systems where logic is used to infer a decision. Thus, an attempt is made to involve this kind of agent in the current research. There does exist, however, an inspiring work by Doelitzscher [62], who proposes an auditing system for IaaS cloud using agent technology. There is a research trend to employ computational intelligence in cloud computing [36] since the cloud service provider can become a suitable candidate for offering security intelligence [43]. Subsequently, in a frequently changing infrastructure, it will be an advantage to deploy an agent-based mechanism [63]. A related work done by [183] to address the use of Artificial Intelligence to mitigate the limitation of the existing security mechanisms such as firewall in protecting cloud computing from the dynamic nature of threats. The utilisation of artificial immune systems in cloud security issues has been studied by [72]

2.9 Summary and Discussion

Infrastructure as a Service (IaaS) is an approach to infrastructure outsourcing. Its flexible characteristics add great benefits in deploying and managing resources from a business perspective. On the other hand, IaaS infrastructure brings with it several security challenges as it changes frequently, is shared among different

customers, enables virtual machine configuration and allows a virtual machine to move easily. As an example, for IaaS operations, live virtual machine migration (LVMM) can be attacked through the network link or through the physical host or even as a result of inconsistent policy configurations.

To mitigate these security challenges, a proper access control model and an enforcement mechanism are essential to enhance IaaS security. In this paper, the firewall as an access control enforcement mechanism is explored and the access control models for cloud computing are investigated. Through this investigation, it has been observed that the firewall faces several limitations in the cloud environment and even the firewall service offered by commercial clouds in form of security group has limited functionalities.

The finding illustrates that cloud firewall system should offer flexibility to the customer in addition to an acceptable level of trust. The virtual firewall as well adds an advantage to IaaS if it is designed and implemented accurately to be aligned with IaaS characteristics. The cloud firewall should also be built on a suitable access control policy to alleviate the security challenges faced by IaaS.

The centralised and the distributed administration approaches for a firewall system offer some useful gains alongside their limitations. We can point out that the distributed administration approach for firewall looks to be more effective than the centralized one in the IaaS environment.

Researchers have proposed several approaches to putting forward a cloud firewall system. However, academic researchers recommend to take access control out of the network and place it in the hypervisor, basically into the host [173, 210, 240].

Moreover, the traditional access control models are not adequate for implementation in the cloud environment. Researchers have proposed several cloud access controls, but still some commercial clouds deploy a classic role-based access control model. Therefore, it is recommended to perform a thorough exploration on the proposed access control to come up with an improved model that is suitable for IaaS and can attract the commercial cloud to deploy it.

We recommend employing an intelligence security approach in implementing and monitoring the access control in IaaS to respond to the challenges faced by traditional firewalls and access control models. Intelligence security is a fertile approach, as most of the existing security paradigms suffer from reactive and fragmented approaches [115]. The cloud service provider may become a convenient candidate for offering security intelligence [43]. In a frequently changing infrastructure, it will be an advantage to deploy an agent-based mechanism [63].

Chapter 3

The Enhanced Attribute-Based Access Control $\mathbf{ABAC_{sh}}$

3.1 Introduction

This chapter addresses the attribute-based access control enhancement in $\mathrm{ABAC_{sh}}$, with a focus on the context-aware mechanism (Section 3.2) and the SoD principle (Section 3.3). This chapter concludes with a computational complexity and quality metrics evaluation of $\mathrm{ABAC_{sh}}$.

3.2 Context-Awareness Mechanism

3.2.1 Overview

The specifications of the context-aware system can be interpreted differently based on the way researchers define a context within their study scope. In [112], context-aware based access control allows dynamic grants for permissions to access objects based on the current user context. The context of the subject and the object can be extracted from the environment by using 5W1H (who, where, what, why, when and how) [111]. In [108], context is considered as a finite set which reflects the system-dependent attributes set by the administrator and differs from the subject attributes and object attributes. However, other authors consider Attribute-Based Access Control (ABAC) to support context-aware features as its access enforcement is based on the attributes of the subject and the attributes of the objects which reflect their context [44]. A context-aware investigation from the literature is explored in the following section and analysed.

3.2.2 Analytical Study of Context-Aware in ABAC

Kim's work [112] investigated context-aware access control of an ubiquitous application with RBAC extension by involving a context-aware agent and state checking matrix. The context-awareness is enforced via two cases. Firstly, the user privilege should change based on the user context, such as location and time. Secondly, the resource permissions changed based on the system information, such as network bandwidth and memory usage. Kim proposed another work related to context-aware access control [111] by presenting a new access control model called CIAAC (Context Information-based Application Access Control), which was designed to separate context awareness and access control policy from processing logic and business, where context is defined as all the information surrounding an entity. CIAAC allows flexible business application operation related to the changes in access control policy to meet external security environment requirements. However, its potential drawbacks have not yet been evaluated. Li proposed in his thesis context-aware attribute-based techniques for data security and access control in mobile cloud environment [123]. Context-aware terminology in his work involves the surrounding context-information of a user as well as location and time. His work is based on Attribute-Based Encryption (ABE) and presents new user-centric access control technique tailored to mobile cloud environments.

The research in this thesis does not consider encryption techniques as ABE faces several limitations. ABE lack an efficient mechanism to support a complete set of comparison relations in policy specification, as authorisation can be complex and attributes can be multi-dimensional [148]. ABE also introduce a heavy computation overhead caused by bilinear pairing, and cannot achieve fine-grained access control [229]. AL Kukhun [7] presents an adaptive situation and context-aware access in pervasive systems. Her proposed model is based on RBAC and makes use of XACML language. There are several attempts to extend RBAC to adopt context-aware features, but these approaches do not take into consideration usability, situation awareness, and improving access opportunities. AL Kukhun discusses several context-aware definitions which can be summarised as a sub-set of conceptual states or information that can be used to characterise the interest or situation of a particular entity. It can be observed that location and time are used as context-aware parameters in most related work on context-aware access control models. Lie and Wang [131] present the Fine-grained Context-aware Access (FCAC) model for Health Care and Life Sciences (HCLS) using specific communication technology based on linked data. FCAC is based on two main components: an ontology base, and access policy with XACML. The researchers defined context attributes as environmental attributes such as network location and time which are

not related to the subject-attributes or object-attributes. Venkatasubramanian *et al* [217] distinguishes between context-aware based on the traditional authorisation models and their proposed criticality-aware based authorisation model. Based on their investigation, the traditional context-aware definition looks only at the subject contextual information, whereas their proposed criticality-based model takes the contextual information about the components of the whole system, not just the subject. The proposed Criticality Aware Access Control (CAAC) is based on RBAC concepts. CAAC evaluates the system context attributes continuously to determine if they have deviated from the norm, and the user privileges are updated before they trigger an access-request. Choi [48] proposes an ontology-based Access Control Model (onto-ACM) that takes context-awareness into consideration. Compared to C-RBAC (Context-aware RBAC), onto-ACM can grant the role inheritance by administrator and user, whereas C-RBAC grants the role by administrator only. Onto-ACM aims to use access-aware technology within a cloud computing environment.

3.2.3 Context-Aware Deployment in IaaS by ABAC$_{sh}$

From prior investigation into the context-aware definition of access control, it can be concluded that the context attributes are those attributes which affect the decision calculation and are not related to the object or subject attributes. As ABAC is the basic conceptual policy selected for IaaS, the entities context is inherited in the form of attributes. Therefore, there is a need to define context-awareness within ABAC.

Context-awareness in our proposed ABAC$_{sh}$ model is deployed in two phases. The first phase defines the context-attribute set. Each context-attribute consists of an attribute name and an attribute value. The context attributes-names set is predefined by the system administrator based on system critical information and characteristics. Context-attributes differ from the environment attributes in that the latter values are predefined by the administrator, whereas context-attribute values are updated based on the system states, where an embedded sensor captures the context information. For example, for the context-aware attribute named memory, its value will be updated based on the system memory measurements. The context attribute can reflect CPU clock, desk space, network zone, or data and time. In the second phase, context-awareness will be defined as one of the configuration points in the proposed ABAC$_{sh}$ system to enforce the use of context in the access-control decision.

3.3 Separation of Duty (SoD) Principle

3.3.1 Overview

In an environment that allows policy combination, a user is authorised to act in more than one role or trigger more than one operation simultaneously. Policy combination might lead to policy conflict, as some actions violate the overall policy if they are committed at the same time. Therefore, constraints should be configured to manage this possibility.

The Separation of Duty (SoD) principle is used in such scenarios to prevent misuse of the system by limiting the user to the least privilege necessary to perform their required tasks. The least privilege principle limits the access of the subject during an operation on a specific task to be within the minimum resources, lowest privileges, and specified period of time. Several security enhancements can be gained from SoD, such as fraud prevention and error minimisation [35, 119, 220].

There are two main types of SoD: static, and dynamic. Static-SoD (SSoD) will list the conflicting roles which cannot be executed by the same user at the same time, whereas dynamic-SoD (DSoD) enforces the control at the time of access-request. In an RBAC model, roles and role relations are defined in advance during the policy engineering process. For SSoD in RBAC, SSoD relations place constraints on the user-to-rule assignment function, where one user can be assigned a specific set of roles and be excluded from another set of roles. Otherwise, two or more users are required to be involved in accomplishing sensitive tasks, since it is less likely that multiple parties will issue a fraud attack. In the DSoD relation, the capabilities for one user are restricted to being activated during a specific user session, i.e. the same user cannot perform two roles simultaneously [76, 153].

Although in RBAC, SSoD and DSoD relations offer some advancement in control over identity-based systems, security issues remain. The most accommodating form of SoD is History based (HSoD). Although, enforcing it in a static based access control management environment such as RBAC is difficult, if not impossible [74, 196]. One role of a HSoD is that it prevents an object from being accessed by the same subject a certain number of times [94]. Therefore, we assume that the ABAC model concept has the characteristics of supporting certain types of SoD. The following section will investigate efforts in the literature to involve the SoD principle in ABAC.

3.3.2 Analytical Study of SoD in ABAC

A significant amount of research has been conducted regarding the principle of Separation of Duty (SoD) in RBAC; however, SoD deployment in ABAC remains

a problem [193]. One of the earliest related works in specifying constraints in ABAC is illustrated through ABACα configuration-points [108]. Nevertheless, their proposed constraint settings are event-specific during attribute assignment and/or modification of the object and subject. This method is similar to the RBAC constraints-setting concept, where the allowed roles are activated for a specific user session after the roles are assigned to the users.

The author of ABCL proposed an event-independent constraint language based on conflicting relations of attribute values such as mutual exclusion and precondition [29]. ABCL language specifies restrictions either on a single set of attribute values or on a set of values of different attributes within the same entity. The usefulness of ABCL language has been validated through case studies. However, it lacks a framework or a formal model that illustrates its implementation.

Dynamic Separation of Duties (DSoD) is more appropriate to cloud computing, and it also meets the dynamic nature of ABAC. Nguyen [152] has carried out interesting research on DSoD and proposed DSoD deployment through Provenance-based Access Control (PBAC). His work is basically proposing a means to capture and utilise the information needed in the SoD enforcement, as previous work in the area assumes that the information is ready without demonstrating how to prepare it. Some of the previous work related to dynamic-based SoD is ObjDSoD, which is based on the object, and where the enforcement is constructed on a set consisting of conflicting-roles and a conflicting-action on these roles. Therefore, the subject will not be allowed to perform an action on an object if that action is in the set of action role conflict. Another approach is OpsDSoD based on operations. This is a task-aware that involves an action-role conflict set, thus it differs from ObjDSoD by limiting the user to perform the needed actions for a particle task even though they have more privileges. A third approach is HDSoD, which combines ObjDSoD and OpsDSoD. Further, HDSoD is object-aware and a task-aware. HDSoD is order-aware, where order-dependency conflict is triggered if the order is essential for a sequence of sub-tasks. Nguyen in [152] extended HDSoD by adding dependence-path-aware and past attribute-aware in their DSoD which is used in Provenance-based Access Control (PBAC).

Event pattern and response relations called obligations are introduced by Ferraiolo, Atluri, and Gavrila in their policy machine research [74], which can enforce some forms of HSoD in their access control framework. Obligations have a set of conditions that are specified by the event pattern under which the state of the policy is obligated to change; only if this set matches the surrounding context, the operation on an object can be executed. There are two recognised standards can be applied to the ABAC concept: Extensible Access Control Markup Language (XACML), and Next Generation Access Control (NGAC) [75]. XACML

does not show any support for DSoD constraint, while NGAC does show some support to DSoD through a Prohibitions (Denies)-relation, which includes a set of denying relations that specify privilege exceptions where a user that is allowed to run capability (x) will be prohibited from running capability (y).

3.3.3 SoD Design and Deployment in Access Control System

It is most likely that the formulation of SoD requirements are prepared by the administrator based on the business rules. An example of such a rule is, person may not approve his or her own purchase order [35]. SoD deployment can be involved with different layers of an access control system. It can be designed within administrative-level policies and procedures, or it can be used within logical or technical mechanism access-control restriction points [119].

Based on recommendations regarding SoD implementation to traverse its limitation in RBAC [196], several techniques have been explored, such as grouping concept, membership control, activation control, history control, and labels. However, in ABAC, the grouping concept will not be appropriate as grouping restricts an attributes flexible nature. Membership control cannot be adopted by ABAC as it is not role-centric. Though, the activation control concept has been adopted into SoD specifications in ABAC by Jin [108] and Bijon [29]. Ferraiolo *et al* [74] describe a relation between entities that can be used in History based SoD deployment. Whereas Biswas *et al* [30] point out that label concept can be used to enforce SoD in their proposed label-based access control in an ABAC.

There are several obstacles in designing and implementing SoD, as it is an application-oriented policy where the business rules indicate the critical tasks which require SoD enforcement. Another challenge is that different applications may require various types of SoD. Lastly, most SoD types are informally defined, which creates ambiguity regarding the subjects or specifications [86].

3.3.4 SoD Deployment in IaaS by ABAC$_{\mathbf{sh}}$

Based on the above investigation [29, 107, 152, 193], SoD can be defined as an enforcement constraint configured to avoid conflict between policies. This conflict can be due to multi-access requests from different subjects to the same resource simultaneously, or the same subject requesting access to multiple resources at the same time. From this definition, it can be observed that SoD may be viewed as object-operation-oriented, which can be aligned with ABAC's relation between object-attributes and operations. We can discern from the above that it is more

appropriate to enhance SoD by implementing a form of HSoD which will be suitable to be enforced in a dynamic access control policy environment such as ABAC.

With RBAC, the centric entity involved in the SoD principle design is the role set. In contrast, ABAC cannot consider a role in the form of an attribute as it can lead to a chaos [52]. Therefore, the focus of this paper regarding formally defining SoD within ABAC will be on attributes and attribute-relations, with no aim to define an application-oriented SoD. Thus, we aim to identify a logical based design for SoD within the ABAC policy model. The proposed work is based on formal logic; exception cases are not encouraged in a formal logic as exceptions make regulations non-monotonic and introduce conflict between proven conclusions [61]. Therefore, the proposed SoD is operation-object orientated that defines a rules-set reflecting the forbidden operations on the set of objects under a specific situation of a collection of entities attributes. Entities include the object, the subject, the environment, and the system context. Moreover, formal logic facilitates SoD rule creation, even by non-expert security administrators. Since the proposed system is attribute-based, it is not necessary to update different locations if a new action-restriction is added, deleted, or modified.

3.4 Evaluating $\mathbf{ABAC_{sh}}$ Computational Complexity and Quality Metrics

3.4.1 Overview

Computational complexity theory studies the resources required to solve a given computing problem. This study can look at the time or/and the space requirements in order to study the system behaviour. Computational complexity of an algorithm can be measured by determining their computation duration and the memory consumption. This task can be accomplished through two approaches.

The first is benchmarking which measures the duration in seconds and memory in bytes for a particular system implementation and input. The second is a mathematical analysis of algorithms which use asymptotic analysis through $\mathcal{O}()$ notation. In contrast to benchmarking, it is independent of the system implementation as different systems can be written in different programming languages, running on different computer specifications, and using different compilers and data-set. Therefore, benchmarking results is difficult to predicate how well the algorithm performs in a different system. In this research, we use $\mathcal{O}()$ notation approach because it balances between precision and ease of analysis. [182, 180]

3.4.2 Evaluation Results

The performance of access control enforcement is crucial for a system that has a large number of users, such as cloud computing. Computational complexity calculations provide an insight to measure performance by including the number of required operations in granting access decisions and safety checks. The existing access control model based on RBAC introduce a high time complexity and colossal space complexity process [211].

This section investigates the computational complexity of three ABAC models: the proposed ABAC$_{\text{sh}}$, the basic ABAC presented by ABACα [108], and the hierarchical ABAC presented by HGABAC [192]. The two primary operations of ABAC models are attribute assignment and policy creation. The safety check is an add-on feature which enhances the access control system. Table 3.1 demonstrates the complexity evaluation between these three ABAC models. Attributes assignment in ABAC$_{\text{sh}}$ is based on the attributes function Att(), which returns the attributes of the required entity. Therefore, it can be represented as a constant function in terms of $\mathcal{O}$-notation, which is also applicable in ABACα. The constant function in terms of $\mathcal{O}$-notation written as $\mathcal{O}(1)$. On the other hand, HGABAC has a group assignment in addition to the attributes assignment because it adopted a hierarchical approach. However, it affects the performance based on the cases implemented, as illustrated by the HGABAC authors in their case-based evolution. In $\mathcal{O}$-notation, HGABAC is represented as $(\mathcal{O}(\mathcal{N}.\mathcal{M}))$ where $\mathcal{N}$ represents the attributes assignment and $\mathcal{M}$ the group assignment. The hierarchy feature is known in RBAC as role-hierarchy which allow an inheritance relation between roles. When a user is assigned to a role, s/he is indirectly associated with the junior-role capabilities. On the other hand, rule hierarchies in ABAC are to add attributes hierarchy to subject and object by adding two type of assignment relations. Subject-attribute to subject-attribute assignment and object-attribute to object-attribute assignment. The advantage of hierarchy is to add attributes Inheritance benefits. That results to fewer rules and reduce relations of the object to object-attribute and subject to subject-attribute [73]. Hierarchy Grouping attribute-based access control (HGABAC) is proposed by [192] with an aim to be able to express DAC, MAC and RBAC. The hierarchy is out of this research scope at this point.

Regarding safety check, only ABAC$_{\text{sh}}$ supports SoD. The complexity of this operation is in $\mathcal{O}$(n) in which SoD rules are consist of **(n)** forbidden rules. The time for this operation will increase proportionally as there will be a loop of size **(n)**. Moreover, ABACα does not provide a logic for policy creation where HGABAC employees truth table schema via K3 logic. As will be explored in Section 4.2,

the most appropriate logic to express access control system is modal logic. Our proposed ABAC$_{\text{sh}}$ expresses policy creation via ACL-Dl which is a type of modal logic as will be de discussed in detail in Chapter 4.

Table 3.1: Complexity evaluation between ABAC models

Model	**attributes assignment**	**policy creation**	**safety check**
ABAC$_{\text{sh}}$	$\mathcal{O}(1)$	Via kripke model using ACL-DL logic	$\mathcal{O}$(n)
ABACα	$\mathcal{O}(1)$	Not included	None
HGABAC	$(\mathcal{O}(\mathcal{N}.\mathcal{M}))$	Via truth table using Kleene K3 logic	None

The quality investigation is based on selected attributes from access-control evaluation quality-metrics suggested by NIST organisation [94, 97, 98]. Table 3.2 summarises the evaluation between ABAC$_{\text{sh}}$, ABACα and HGABAC. ABAC$_{\text{sh}}$ is context-aware and supports operation-object oriented Separation of Duty (SoD), whereas ABACα and HGABAC do not. In general, attributes-engineering is not explored by any of those ABAC models, but in terms of the attribute-object assignment, ABAC$_{\text{sh}}$ creates a capability list using deontic operator obligatory per object. ABACα uses two configuration points in assigning attributes to objects; the first is an object attribute assignment constraint at object creation time, and the second is an object attributes modification constraint. HGABAC has a total of seven mapping functions: direct, consolidate, member, inherited, effective, name, and parents. These mapping functions support hierarchical feature to ABAC, yet it increases the complexity of the model by adding additional operations related to grouping. Therefore, HGABAC has the highest performance complexity among these three ABAC models. ABAC$_{\text{sh}}$ performance complexity is higher than ABACα as it adds extra operations for SoD checks. Finally, ABAC $_{\text{sh}}$ is formally described and verified via a complete formal policy language based on modal logic, which is ACL-DL. ABACα is missing a complete formal policy language, whereas HGABAC is based on propositional logic which is more complicated in terms of computing than ABAC$_{\text{sh}}$

3.5 Summary and Discussion

This chapter introduces the characteristics that contribute to enhancing the ABAC model.

From the prior investigation into the context-aware definition of access control, it can be concluded that the context attributes are those attributes which affect

Table 3.2: Quality metrics evaluation between some ABAC models

	ABACα	**HGABAC**	**ABAC$_{\mathbf{sh}}$**
Support for SoD	-	-	✓
Context-aware	-	-	✓
The steps required for assigning attributes to object during access control entries creation	Via two configuration points	Via mappings function total of seven	Via capability list
The number of relationship required to create an access control policy	Not specified	Total of four relations	Total of two
Performance complexity of access control enforcement	Low	High	Moderate
Syntactic and semantic support for specifying AC rules	Syntactic only	Language based on propositional logic	Language based on modal logic

the decision calculation and are not related to the object or subject attributes. In ABAC$_{\text{sh}}$, the context-aware are not pre-defined values. Instead, context attributes reflect the system state values during the access request process. This consideration meets the IaaS scalability feature.

Furthermore, this chapter clarifies SoD principles and deployment methods. The proposed SoD deployment for IaaS is a type of operation-object orientated. This SoD defines a rules-set reflecting the forbidden operations on the set of objects under a specific situation of a collection of entities attributes. Entities include the object, the subject, the environment, and the system context. Since the proposed system is attribute-based, it is not necessary to update different locations if a new action-restriction is added, deleted, or modified. That support the dynamic feature of IaaS.

Finally, a comparative study evaluation illustrates how ABAC$_{\text{sh}}$ is distinguished from existing ABAC models. the Evaluation of the ABAC$_{\text{sh}}$ computation complexity and access control quality metrics reveals superior results in comparison with two existing ABAC models.

Chapter 4

The Proposed ACL-DL Logic for $ABAC_{sh}$

4.1 Introduction

The aim of this chapter is to present a formal logic that is able to specify $ABAC_{sh}$. Being attributes centric is the main different between ABAC models and other access control categories. The attributes of access-control entities are used to make access decision; whereas other access control categories are using identities values. The authentication mechanism is responsible about checking the users identities, therefore the authorization mechanism through ABAC model focuses on more fine grained access.

The formal logic approach is selected in this thesis because it supports access control system specification, verification and reasoning as will be analysed in Section 4.2. Access Control Logic (ACL) was investigated in the literature by different scholars who contribute into clarifying its features as discussed in Section 2.7.3. However, ACL faces a problem of implicit permission because the access permission is based on entailment through the binary modality ***says***. The format of ***says*** involve the use of subject identity. For example, ***A says*** ϕ interpreted as a principal **A** support a formula ϕ to hold in the system. In another word, whenever ϕ is true ***A says*** ϕ is true as well. Moreover, ACL support the primary access control categories. For example, it needs to be extended with additional relations to specify RBAC model characteristics. To the best knowledge of the author, no prior work introduces ACL enhancement for ABAC.

Based on the presented literature, the proposed logic in this research will combine Access Control Logic (ACL) modality ***says*** with a Deontic Logic (DL) modality ***obligation***. ABAC is based on attributes rather than subject identity; therefore, the problem of implicit permission is eliminated. Deontic Logic is se-

lected in this research because of its capability to express access control rules as discussed in Section 2.7.2. The contribution of this study in access control logic research is illustrated in Section 4.3 .

In this research, the understanding of modal logic and background are based on [47, 214, 84, 83, 31, 234, 101, 32, 174]. The production of the proposed logic Access Control Logic-Deontic Logic (ACL-DL) is divided into two steps, as illustrated in Figure 4.1. The first step involves an inductive definition of a formal language to produce a well-formed formula in addition to logical-rules and axioms. The second step is to define a model that is able to interpret the language sentences. The model is based on a Kripke structure that allows the definition of the truth notation in model-theory and to define the logical truth valuation within the scope of ABAC properties.

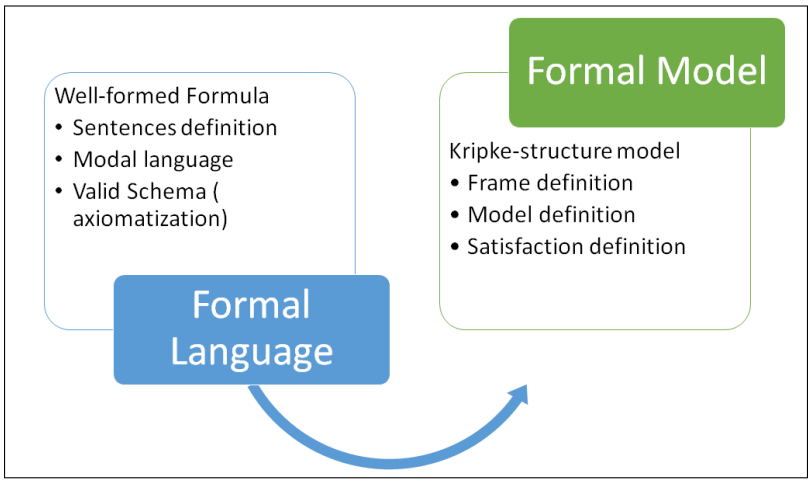

Figure 4.1: ABAC formal Modelling steps

Formal languages in modal logic are defined by the induction approach that starts from a specific observation to reach the hypotheses and/or theorems, contrary to the deductive approach. Induction is less restricted than deduction, even though the former is considered logically stronger than the latter. Nevertheless, in some cases, conclusions can be true in terms of format via deductive reasoning while being false in terms of the content. Each research area selects the technique that suits their used language and scenarios. This research uses reasoning within a finite model, which is a closed world with internal relations. Therefore definitions, proof, and reasoning by induction are suitable in this scenario where the modal logic is considering the relational structure within the local scope. Modal formulas are evaluated from inside the structure at a particular state.

4.2 Analytical Study: Why Modal Logic is Suitable to be Used in IaaS Access Control?

Formal logic language is one of the most important aspects of policy specification [16]. From an administrative point of view, logical policy-based access control models are significantly expressive. Therefore, they add flexibility and provide an opportunity to present a variety of authorisation policies. For example, the U.S military faces a problem with access control due to frequent large-scale changes to its authorisation policy, thus, as pointed out by [93], a logical access control system could solve this problem. IaaS cloud is dynamic, therefore there is a frequent change in access control rules, so, logical access control is more appropriate, as explained in Figure. 4.2.

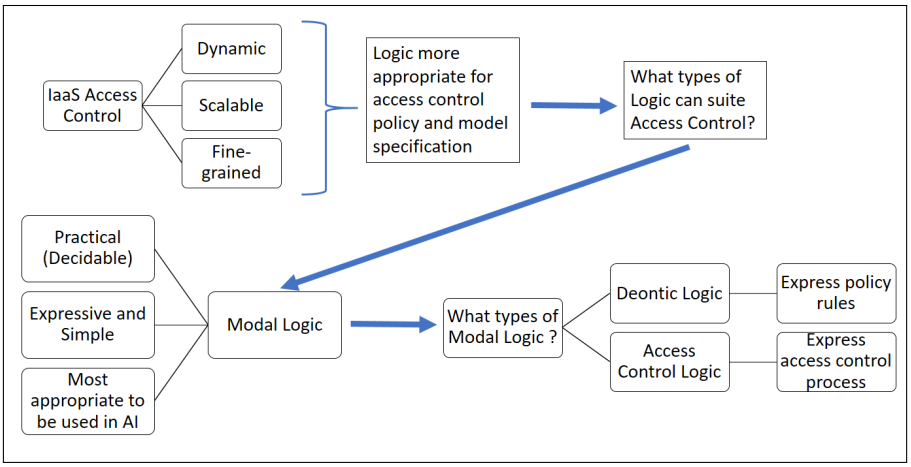

Figure 4.2: Why Modal Logic is Used in IaaS Access Control

Access control policy has been investigated for more than forty years [55]. The first logical specification for access control policies was based on first-order logic, where permission is directly assigned to a subject, but then a role concept is suggested, as in RBAC where permission is assigned to a role instead of a specific subject. Consequently, the role concept has been modulated using predicate, but there is a lack of formalisation, as RBAC does not have any formal semantics. The lack of formal semantic leads to the consideration of the use of role context-dependencies, such as location and temporal. Therefore, there is a research direction that takes into account the involvement of deontic logic and context-aware concepts. Consequently, modal logic has been selected in this research to specify an enhanced ABAC model.

Modal logic is a simple yet expressive language for talking about relational structures. Modal language provides an internal, local perspective on the relational structure. This feature makes it unique among other forms of logic. Modal

formulas are evaluated inside a structure at a particular state [31]. This characteristic of modal logic, introduce an advantage in specifying ABAC that supports the dynamic features as ABAC decision state is vary based on the provided attributes.

Abadi [2] points out that the core argument in considering the proper type of formal logic to be used in an access control model is based on the ability of this logic to write formulas that meet the model requirements. Therefore, this thesis has found that modal logic is the most suitable logic that satisfies the ABAC model requirements in IaaS cloud.Even though modal logic is effective in access control systems, it was not used in practice to enforce authorisation policies even though modal logic has proved to be used in theoretical access control, as elaborated by Genovese [83]. Thus, this thesis aims to provide a practical framework for the proposed ACL-DL based on AI architecture, as Modal Logic is a serious candidate that supports the logical approach in artificial intelligence systems [102].

A modal logic approach based on deontics is more suitable for expressing access control than classical logic. Based on the semantic, the core modalities of modal logic are possibility and necessity. Deontic logic is the logic of obligation, permission, and forbidden. It adds intentional connectors to the classical Boolean connectors [116, 142, 202].

Moreover, dealing with conflicting rules is one of the major limitations of classical logic in policy specification of access control systems. In a classical logic such as first-order logic, which is also called predicate logic [102], and its related approaches, every conclusion can be drawn from a contradictory set of premises, which leads to trivialisation. This effect is not acceptable for access control practical purposes [16]. The conflicting rules limitation adds another advantage of replacing classical logic with deontic logic.

4.3 The Contribution of This Study to Access Control Logic Research

Several studies on access-control logic have been proposed, mostly related to policy representation, enforcement, and proof-theory. The literature contains limited work that explores decidability, Kripke semantics, and the relation between the logics as we agree with [82] observations. The proposed ACL-DL logic in this chapter is able to represent policy rules. It has a defined Kripke semantic, in addition to the relations between the logics.The ***says*** operator is combined with deontic Logic operators (DL) to introduce a modal logic that is able to specify and reason an ABAC system. DL facilitates security policy rule specification, where ***says*** gives a logical relation among access control system entities.

Three well-known policy-oriented logics are used in access control systems: first-order logic, stratified logic, and DL [42, 57]. Based on this research, only DL is a branch of the modal logic family. Thus, it has been involved in our proposed access control language. In order for DL to be practical in computer science, many researchers recommend incorporating it with other modal logic modalities [38]. The proposed logic incorporates ***says*** modality with deontic modalities to create a logic that is able to formally express attributes based access control.

In current access control logics, permission through entailment is defined where binary modality ***says*** is used. Because the identity of the principal is not included in this modality, a problem of implicit permission occurs [83]. In the proposed ACL-DL language, by default, the principal identity is not included. Instead, a set of attributes and a policy sentence are used to give access permission since the proposed ABAC model is identity free. Therefore, this limitation is overcome through the use of a ***says*** operator.

However, defining compelling semantics for ***says*** is an open problem in access control logics [61]. This problem is beyond the current research dimension. Deontic Logic (DL) paradox research is also beyond the scope of this research. An attempt is made by this research to partially solve the DL paradox by allowing a relation between principals [45]. This relation is achieved through the variable sharing principle. The basic idea is to have at least one shared propositional variable in common between the premises and the conclusion during the reasoning process. This variable sharing condition is met in the proposed ACL-DL, as the access control request has a propositional variable that reflects the requested action (operation on the object) where the conclusion reflects the access decision, which indicates that this action is allowed or denied. The next section will demonstrate a formal verification for the proposed ACL-DL logic.

4.4 The Formal Logic Language

Formal languages in modal logic are defined by induction where its start from specific observation to reach the hypotheses and/or theorems, unlike deductive definition. Induction is less restricted than deduction, even though the former considered logically stronger than the latter. Nevertheless, in some cases, some conclusions are true in term of format via deductive reasoning while it is false in term of the content. Each research area selects the proper technique that meets their used language and scenarios. In our research, we are reasoning within a finite model which is a closed world and has internal relations. Therefore definitions, proofs and reasoning by induction is suitable in this scenario where the modal logic is considering the relational structure within the model locally scope. Modal

formulas are evaluated from inside the structure at a particular state.

4.4.1 Syntax

The syntax of a formal language shows what the formula looks like. A formula is constructed by connecting an atomic proposition using connectives. The atomic proposition is a statement or assertion that must be true or false where connectives are such ($AND = \wedge, OR = \vee, not = \neg$). The syntax of access-control logic consists of formulas and principle. The formula represents a basic statement that consists of an action on an object such as read file foo. The principle represents entities that make the statement, such as people, machine, process, which is mainly the subject part of the access-control system. The statements can be primitive and consist of one formula, or can be compound, with more than one formula. The principle can also be a simple or compound.

4.4.2 Language Operators

The proposed ACL-DL language will make use of the logical operators listed in Table 4.1, which consist of a well-known access control operator ***says***, and two main deontic logic operators: obligatory OB, and permissible ***PE***. Deontic operators reflect the juridical notation of an access control system, whereas ***says*** reflects an access-request notation. ***OB*** and ***PE*** are unary modalities where ***says*** is a binary modality. ***OB*** and ***PE*** are dual pairs, as there are many notations in natural languages where ***OB*** is aligned to necessity and ***PE*** is aligned to possibility. Necessity and possibility are the modal-logic basic operators. The ***says*** operator works similarly to how an epistemic-logic operator believes with limited binary relation of type serial.The representation of a subject (S) requesting an operation (read) on an object (file_x) can be expressed formally in the form S ***says*** (read file_x). At this point, the subject triggers an access-request as they believe that they can obtain access. This access-request is not granted unless there is a policy rule indicating that the subject is capable of this action. Capability indicates the access right and can be expressed formally in the form ***OB*** (S ***says*** (read file_x)), the access-request is then granted.

In the literature, there are other access-control logic operators such as controls, specks-for, hands-over, and servers, which are not involved in this research. However, the combination of the deontic operator obligatory (***OB***) with the access control operator ***says*** in ACL-DL logic is facilitating rule expression.

Deontic operators have a variety of presentations, such as permissible(***PE***), obligatory(***OB***), optional(OP), impermissible(IM), and omissible(OM) [205, 84, 18]. These operators can be derived from ***OB***, as illustrated in Table 4.2. Therefore,

Table 4.1: AC-DRL operators with intended informal meaning

OB: obligatory to access
PE: Permissible to access
Says: a principle P make the statement Q, P make statements that they believe(conceive) to be true

deontic logic operators give the flexibility to accommodate different policy-rules formations.

Table 4.2: Deontic operators derivation

OB(p)obligatory is a central term
IM(p) = OB(¬ p), not allowed (impermissible or forbidden)
PE(p) = ¬ OB(¬ p), allowed (permissible)
OM(p) = ¬ OB(p), you don not have to do it (omissible)
OP(p) = ¬ OB(p) & ¬ OB(¬ p), you do not have to do it but you can do it (optional)

4.4.3 Principals Expressions Definition

The set of principal expressions is ranged over **P** and **Q** and follow the syntax in Table 4.3, which is expressed in BNF grammar format. BNF (Backus-Naur Form) is a commonly-used type of context-free grammar employed in computer science [180]. The symbol (&) represents a principle in conjunction with another principle. **A** is a simple principal name which represents a subject that creates an access-request, where P and Q represent sets of attributes that are associated with one of four entities: subject-attributes **Att(s)**, object-attributes **Att(o)**, environment-attributes **Att(e)**, and context-attributes **Att(c)**. The symbol (|) means **P** is quoting **Q**, which will be useful when the set of attributes is quoting the subject that makes the access-request.

Example. An example for simple principal **A** is the subject itself such as a user of type student. An examples for set of attributes P are a collection of four attributes sets: **Att(s)** such as the student is a PhD from computer science department, **Att(o)** such as the Student wants to access a resources for computer science students that are related to post-graduate courses, **Att(e)** such as this operation is allowed for school of science and the student registration is valid, **Att(c)** such as the time should be in the morning and at semester time.

Table 4.3: principle syntax expression

$P ::= A \mid P\&Q \mid P \mid Q$

4.4.4 Formulas Definition

Formula definition requires the definition of a class of general modal language that is relativised to a set of atomic formulas. Let **VAR** be a non-empty finite set of atomic formulas which known as well as primitive propositions. The members of **VAR** set can be written as **VAR=**$\{$ **p1,p2,,pn**$\}$ and for simplicity, atomic formulas are denoted **p, q and r** to range over **VAR** elements. The smallest set of formulas is ϕ of the modal language **L** are defined in BNF format in Table 4.4.

Examples. A simple operation on an object such as restarting a virtual machine is denoted as **p**, whereas $p \in VAR$. A combination of several atomic formulas using boolean operators and the language operators discussed in Section 4.4.2 will lead to a combined formulas such as a set of attributes are required to access an object $S|(Att(s)\&Att(o)\ \&Att(e)\&Att(c))$ says $(opr \rightarrow obj)$.

Table 4.4: ACL formula syntax

All propositional formulas
$p \in VAR$ where p is an atomic formula
$VAR \subseteq \phi$
$\phi ::= VAR \mid \neg\phi \mid \phi \bigwedge \varphi \mid \phi \bigvee \varphi \mid \phi \rightarrow \varphi \mid \phi \equiv \varphi \mid Psays\phi \mid OB(\phi)$

Finite model in term of Sub-formulas Definition

Defining sub-formulas is useful as a modern finite version has been observed to be the standard in the literature. Many proofs, such as proof of completeness, can be formed in a finite universe of formulas. Therefore, the definition of a finite model consists of formulas where the model size is a function of the number of sub-formulas which can aid the decidable conclusions of modal logics. Therefore, 'φ sub-formula of ϕ is defined in 4.5.

Table 4.5: ACL Sub-formula syntax

φ is a subformula of itself and all the formulas used to build φ
if $\psi \rightarrow \neg\varphi, \varphi \rightarrow \phi$,or$OB(\varphi)$ then $\varphi(\phi)$ is a subformula of ψ
if φ is a subformula of ϕ,and $\phi$is a subformula of ψ ,then φ is a subformula of ψ
$T = \neg \bot$
$\varphi \bigvee \psi = \neg(\neg\varphi \bigwedge \neg\psi)$
$\varphi \rightarrow \psi = \neg\varphi \bigvee \psi$
$OB(\psi) = \neg PE(\neg\psi)$
$Psays\psi = \psi$

4.5 The Formal Model Structure

4.5.1 Background

Relational structure is used to explicate the logical structure of modal systems. It consists of a collection of relations on a defined set. The relational structure can be observed in various domains. Mathematical structures can be expressed as a form of relational structures. Moreover, diverse applications discern to be a relational structure, such as the use of labelled transition systems to model program execution in theoretical computer science, in addition to disciplines such as knowledge representation, computational linguistics, formal semantics, economics, philosophy, information security, artificial intelligence, and others.

In modal language, a relational structure is called a frame that is a pair $\mathcal{F} = (\mathcal{W}, \mathcal{R})$,where $\mathcal{W}$ is a non-empty set and $\mathcal{R}$ is a binary relation on $\mathcal{W}$. $\mathcal{W}$ is called the universe (or domain) of $\mathcal{F}$. The elements of $\mathcal{W}$ can hold different names to reflect the modelled system, such as points, states, nodes, times, instants, situations, and others. Binary relations can be serial, reflexive, transitive, symmetric, convergent and more, depending on the axiom needed to represent the system state relations. The model for basic language is a pair $\mathcal{M} = (\mathcal{F}, \mathcal{V})$where $\mathcal{F}$ is a frame and $\mathcal{V}$ is an assigning function, also known as a valuation function that maps each proposition letter $\mathbf{p}$ in a logical formula ϕ in $\mathcal{W}$ to form a subset $\mathcal{V}(p)$ of $\mathcal{W}$. Formally $\mathcal{V} : \phi \rightarrow \mathcal{P}(\mathcal{W})$ where $\mathcal{P}(\mathcal{W})$ denotes the power set of $\mathcal{W}$. Informally, $\mathcal{V}(p)$ is a set of points in the proposed model where $\mathbf{p}$ is true. Given a model $\mathcal{M} = (\mathcal{F}, \mathcal{V})$ denote that $\mathcal{M}$ is based on the frame $\mathcal{F}$ and can be written in the form $(\mathcal{M}, \mathcal{R}, \mathcal{V}(p), \mathcal{V}(q), \dots)$. Therefore, the model is a relational structure consisting of a domain, a single binary relation, and the unary relations given to use by $\mathcal{V}(p)$. Both frame and model are considered relational structures. The frame level is used to make the fundamental assumptions mathematically precise, and the model level is used to add additional descriptive content.

The possible world $\mathcal{W}$ can reflect a variety of system-state elements; it can contain sentences and numbers, even a combination of elements. The critical points in designing a model of possible worlds are to define a finite set of world elements and to decide on the relationships between those elements. The Kripke structure allows the modelling of logic with more than a single valuation. Its valuation mechanisms can be divided into two streams: facts presentation, and actual possibility presentation. The first valuation represents an actual truth value of an atom through a distinguished state in a $\mathcal{W}$ set. This valuation is accomplished through the labelling function $\mathcal{V}$. The second valuation represents an actual possibility through relation R that represents the relative possibility of

the truth value [78].

4.5.2 Semantic Definition

If two logics are combined, such as $\mathcal{L}1$ and $\mathcal{L}2$, if $\mathcal{L}1$ is stronger than $\mathcal{L}2$ based on the type of axioms applied, then the Kripke-structure model for such a language combination should obey the restriction illustrated in Table 4.6 [101]. Consequently, there is a necessity to define an accessibility relation for each operator. The proposed ACL-DL is a multi-modal logic that combines a ***says*** access-control operator with an ***obligatory*** deontic operator. Hence, the axiom followed by ACL-DL operators are of type serial. Therefore, the two accessibility relations of ACL-DL are of the same level of strength.

Table 4.6: Multimodal Kripke-structure restriction

$\mathcal{V}(\mathcal{L}_{\mathrm{n}}\phi, w) = 1$, if $\mathcal{V}(\phi, w) = 1$ for every w' such that $w\mathcal{R}_{\mathrm{n}}$w', and 0 otherwise

A multi-modal system has complex modalities as they are obtained by composing modal operators of different modal logic in order to capture several aspects simultaneously. As a result, multi-modality allows different means of reasoning and different means of interaction during the design of an agent situation [42]. Therefore, multi-modal ACL-DL logic has two frames which can capture two system states: access-request, and rule checking. Its two accessibility relations, $\mathcal{R}_1, \mathcal{R}_2$ follow a serial binary relation, as illustrated in Table 4.7.

Table 4.7: ACL-DL model relation properties

$\mathcal{R}$ is **serial** if for every x in $\mathcal{W}$, there is some y in $\mathcal{W}$, x$\mathcal{R}$y

The ACL-DL first frame $\mathcal{F}_1 = (\mathcal{W}, \mathcal{R}_1)$, $\mathcal{W}$ consists of the states related to access-request preparations using a ***says*** operator. The second frame $\mathcal{F}_2 = (\mathcal{W}, \mathcal{R}_2)$, $\mathcal{W}$ consists of the conceivable states where ABAC rules are checked. $\mathcal{R}_2$ is the relation held between the two states **w** and **w'** if all the rules which **r** establish in **w** are also followed in **w'**. The followed access control rules are denoted by **r**. However, different states might have a different set of rules based on the system requirements. As will be explained later in the $ABAC_{sh}$ formal description, we have two sets of rules: policy rules, and SoD rules.

Definition (Kripke Structure) A Kripke structure of a given a language $\mathcal{L}$ with finite alphabet Σ is a finite structure $\mathcal{M}$ that is ordered quadruple interoperated in Table 4.8.

The meaning of accessibility relations in the proposed language is based on the used modalities which are listed in Table 4.9. A specific reading $OB\phi$ in access

Table 4.8: ACL-DL semantic

Model $\mathcal{M} = (\mathcal{W}, \mathcal{R}_1, \mathcal{R}_2, \mathcal{V})$ where p is an atomic proposition
$\mathcal{M}$ is a non-empty set of words or states
$\mathcal{R}_1$ is a binary relation which make the frame $\mathcal{W}, \mathcal{R}_1$ (W,R1) as an ordered set, $\mathcal{R}_1 \subseteq \mathcal{W} \times \mathcal{W}$
$\mathcal{R}_2$ is a binary relation which reflect deontic property $\mathcal{R}_2 \subseteq \mathcal{W} \times \mathcal{W}$
$\mathcal{V}$ is unary relation which map $\mathcal{W} \times \mathcal{VAR}$ to assign true values and referred to it as labelling function defined as $\mathcal{V} : \mathcal{VAR} \rightarrow 2^{\mathrm{w}}$, so that $\mathcal{V}(p)$ is the set of states where p is true

control rules(r) is defined, as in light of r, it must be that ϕ or in light of r, ϕ is obligatory.

Examples. an example for $\mathcal{W}$ is a set of access control stages such as access-request and access-decision. An example for $\mathcal{R}$ is an access rule to forbidden an action such as an action of **p = restart virtual machine** which can be expressed as **FB(p)**. An example for evaluation function $\mathcal{V}$ is to set a true value for a specific access-request such as it is permissible to restart a virtual machine **PE(p)**.

Table 4.9: modality reading for defined relation in M

$Psays\phi$: principal P believes that ϕ, then v$\mathcal{R}_1$w: w should be the actual world according to the P access-request at v
$OB\phi$: It ought to be that ϕ, then v$\mathcal{R}_2$w: w is an acceptable state according to the information at v
$PE\phi$: It is permitted to be that ϕ, then $\mathcal{R}_2$: v state consider the information given by w state in order to make a permission decision

4.5.3 Satisfaction Definition

Satisfaction Definition

A satisfaction relation denoted by ($\models$) defines when the formulas ϕ is satisfied (or true) in the model $\mathcal{M} = (\mathcal{W}, \mathcal{R}_1, \mathcal{R}_2, \mathcal{V})$ at state **w**, where $p \in \mathcal{VAR}$, $\mathcal{VAR} \subseteq \phi$, $v, w \in \mathcal{W}$. Table 4.10 illustrates satisfaction relation for our ACL-DL language, satisfaction relation is also known as entailment relation. Model $\mathcal{M}$ is said to be satisfied if formula is satisfied by all the model states (worlds). Thus, the satisfaction definition can be written as $\mathcal{M} \models \phi$, if and only if for each $w \in \mathcal{W}$, there is $w \models \phi$.

4.5.4 Axiom

Axiomatisation is the process followed to identify an axiom. An axiom is also known as a schema, while some researchers refer to them as classes. An axiom is a set of sentences which have been proven true in all modal logics and have the same

Table 4.10: Satisfaction Definition

1.$\mathcal{M}, w \models T$
2.$\mathcal{M}, w \models \perp$
3.$\mathcal{M}, w \models p iff p \in \mathcal{V}(w)$
4.$\mathcal{M}, w \models \phi \bigwedge \psi$ iff both $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$
5.$\mathcal{M}, w \models \neg\phi$ iff $\mathcal{M}, w \models \phi$
6.$\mathcal{M}, w \models Psays\phi$ iff for all v such that w$\mathcal{R}$v and $\mathcal{M}, v \models \phi$
7.$\mathcal{M}, w \models OB\phi$ iff $\mathcal{M}, v \models \phi$ for all v such that w$\mathcal{R}$v
8.$\mathcal{M}, w \models PE\phi$ iff $\mathcal{M}, v \models \phi$ for all v such that w$\mathcal{R}$v and $\mathcal{M}, v \models \phi$

form regardless of the used language. Further, some axioms can be true in a specific branch of modal logic, but not under other types, based on their accessibility relation properties. Two axioms are sound and complete for all models: K-axiom, and N-axiom. D-axiom is complete and sound in deontic logic. Unit-axiom is a well-known axiom in access control logic and means that for every true formula ϕ is supported by every principal. Table 4.11 lists the satisfied axioms (denoted with A) and rules of proof (denoted with R) in the proposed ACL-DL language. Since ACL-DL logic extends propositional logic with additional modalities, all axioms and rules which are satisfied by propositional logic are also satisfied by ACL-DL logic.

Table 4.11: ACL-DL Axiomatization

A0. all axioms of the propositional logic
A1. $OB(\phi \rightarrow \psi) \rightarrow (OB\phi \rightarrow OB\psi)$ **(K-axiom)**
A2. $Asays(\phi \rightarrow \psi) \rightarrow (Asays\phi \rightarrow Asays\psi)$ **(K-axiom)**
A3. $OB\phi \rightarrow \neg OB\neg\phi$ **(D-axiom)**
A4. $Asyas\phi \rightarrow \neg(Asays\neg\phi)$ **(D-axiom)**
A5. $\phi \rightarrow (Asays\phi)$ **(unit-axiom)**
R0. All rules of propositional logic
R1. if $\vdash \phi$ then, $\vdash Asays\phi$ **(N-axiom)**
R2. if $\vdash \phi$ then, $\vdash OB\phi$ **(N-axiom)**
R3. if $\vdash \phi$ and $\vdash \phi \rightarrow \psi$ then, $\vdash \psi$ **(Modus Ponens)**

4.5.5 Validity, Soundness, and Completeness

Definition (Validity). A formula ϕ of modal logic is valid if it is true in every state of every model $\mathcal{M}$ is written as $\models \phi$.

Definition (Soundness). Let $\mathcal{S}$ be a class of frames. A normal modal logic $\mathcal{M}$ is sound with respect to $\mathcal{S}$ if $\mathcal{M} \subseteq \mathcal{S}$. In other words, $\mathcal{M}$ is sound with respect to $\mathcal{S}$ if for all formulas ϕ, and all structures $\Omega \in \mathcal{S}$, $\vdash_{\mathrm{N}} \phi \rightarrow \Omega \models \phi$. If $\mathcal{M}$ is sound with respect to $\mathcal{S}$, it can be said that $\mathcal{S}$ is a class of frames for $\mathcal{M}$.

Definition (Completeness). Let $\mathcal{S}$ be a class of frames. A logical $\mathcal{M}$ is strongly

complete with respect to $\mathcal{S}$ if for any set of formulas $\lambda \cup \{\phi\}$, if $\lambda \models_S \phi$. That is, if λ semantically entails ϕ on $\mathcal{S}$ then ϕ is $\mathcal{M}$-deducible from λ.

The formula ϕ is valid on ACL-DL frames only if the relations between the worlds in these frames are serial. ACL-DL soundness and completeness are satisfied within K-axiom, N-axiom and D-axiom for all its frames. Also, access-request frame $\mathcal{F}_1$ is sound and complete within the unit-axiom. Another method to validate the model is through valuation function $\mathcal{V}$, by assigning valid formulas to certain frames. Function $\mathcal{V}$ can be defined based on the system requirements to indicate which formula is valid in a specific scenario that consists of states that have relations without introducing any new world elements.

4.6 ACL-DL Formal Verification

In this section, the formal verification understanding and background are based on [11, 32, 51, 102, 214, 216]. Formal verification for a formally designed system is necessary to provide a mathematical justification that the model does what it is designed to do. It is widely used in the hardware and software industries. Therefore there is growing demand for techniques and professionals to apply them.

Formal verification can be calculated by a human through proof-theory. However, the formal logic modelled through relational structures such as the Kripke-structure have an advantage in their ability to be verified through automated formal verification such as model checking approaches. As the systems become more complicated, there is a need for an automated formal verification. Model checking based verification is simpler then proof based, as the latter verifies an infinite classes of models. Whereas the model checking approach can be used to verify the system property and is classified by the system engineering as post-development methodology.

In the proposed research, a mode-checking approach is used to verify if the proposed ACL-DL model $\mathcal{M}$ satisfies ϕ (written $\mathcal{M} \models \phi$). However, this model-checking computation can be achieved if the ACL-DL model meets finite-model properties.

4.6.1 Finite Model

Finite model theory studies the expressive power of finite models. Most classical models focus on infinite structures, whereas in computer science the objects are finite. Hence, this research concentrated on logic over the finite structure. ACL-DL is described through a Kripke-structure model which is a type of finite structures model. Therefore, the ACL-DL model is a finite model and can be verified through

finite-model checking approaches. Based on the model defined in Section 4.5, the below features are met, hence making it a finite model:

- The set of propositional variables is a finite set, as its elements are countable
- The set of logical symbols are also a finite set
- The set of axioms and the set of models are recursively enumerable

4.6.2 Decidability Verification

Different methods can be used to check the decidability of modal logic, such as finite model, filtration method, tableau system by using decision producer, and normal system obtained from axioms D,T,B,4, 5.

Definition (Normal Modal Logic). Modal logic is said to be normal if its sets of formulas are closed under conditions of modus ponens, generalisation and uniform substitution [32].

ACL-DL is a normal modal logic, as its tautologies satisfy the K-axiom by default since it also satisfies the D-axiom. ACL-DL also has a dual relation and is closed under modus ponens, uniform substitution (N-axiom) and generalisation (unit axiom). Normal modal logic properties allow syntactical validation and conclusion drawing while semantical validations and local consequences are managed through the relational structure. As a result, ACL-DL is decidable as it is a normal and finite model, hence decidability is not a measurement of the complexity, but it is rather an indication that the logic is computable. In order to calculate the complexity of ACL-DL language, it is necessary to verify its satisfiability problem.

4.6.3 Satisfiability Verification

Theorem(Satisfiability). In Section 4.5, every formula ϕ in the defined ACL-DL modal language satisfies the proposed ACL-DL formal model based on a finite tree of depth at most md(ϕ).

Proof. md(ϕ) stands for a model depth that reflects the maximum length of a nested sequence of modality operators. A finite branch tree can be obtained where model $\mathcal{M}$ satisfies ϕ at its root. Since ACL-DL is a finite model, $\mathcal{M}$ satisfies ϕ on a shallow tree. There exists a PSPACE algorithm as the shallow trees branch construction is calculated one branch at a time, where the branch length is bounded by md(), which makes this processing polynomial in the size of the input formula. The processing space (memory) is freed before the next branch is constructed.

4.6.4 ACL-DL Model Checking

As ACL-DL is modelled on a Kripke structure, it is apparent that a model-checking approach that takes the state and the path under consideration is required. ACL-DL is a finite model, therefore, program verification approaches are avoided as model checking approaches are more convenient for finite state models.

Theorem. The model checking problem for ACL-DL logic can be solved in time $\mathcal{O}(\|\mathcal{M}\|.\|\phi\|)$, which is a Ptime complete.

Proof. Based on the proposed language in Section 4.4, a formula ϕ and a Kripke structure $\mathcal{M}$ are defined for ACL-DL logic. If all sub-formulae $\phi_1, \phi_2, \ldots \phi_{\mathrm{k}} of \phi$ are listed in order such that if ϕ_{i} is a sub-formula of ϕ_{j} then $j < i$, there exists an algorithm that inductively labels each state of w (where $w \in \mathcal{W}$ possible world set in $\mathcal{M}$ with ϕ_{i} or $\neg\phi_{\mathrm{i}}$ based on which formula holds in that state. As some states are already labelled via a $\mathcal{V}$ labelling function (also known as valuation function) for the atomic propositional statements ($p \in VAR, VAR \subseteq \phi$). Only for some cases where modality operators are involved such as obligatory operator $\phi_{\mathrm{i}} \equiv OB\phi_{\mathrm{i}}$ for some $j <= i$. In this case for each state w, all states w that have an accessibility relation such wRw are checked, and it is determined whether w has a ϕ_{i} label in the jth step, otherwise it will be labelled as $\neg\phi_{\mathrm{j}}$. Therefore, this algorithm can be implemented in $\mathcal{O}(\|\mathcal{M}\|.\|\phi\|)$ time. From a data structure point of view, this presents an algorithm that consists of a sequence of statements and if-then-else blocks.

4.6.5 Conclusion

The ACL-DL satisfaction problem is PSPACE solvable based on a modal depth checking process. The satisfaction problem in ACL-DL concerns the polynomial amount of space. Formal verification through model-checking is a PTIME as it is solvable in a polynomial amount of time. Therefore, it is moderate in terms of time complexity. Moreover, ACL-DL is a decidable logic, which indicates that it is possible to implement it in a computing machine, which gives it an advantage over other theory-based logics. This feature allows the linking of the benefit of the theoretical computer science with the practical implementation.

4.7 ACL-DL Logic Complexity Profile Evaluation

The ACL-DL complexity profile is a sub-type of simple modal logic; therefore, it has some similarities in terms of the modal-checking problem and the satisfiability

problem. Table 4.12 compares two logic complexity metrics: model-checking, and satisfiability between ACL-DL Logic, Propositional Logic, and First-Order Logic.

Table 4.12: $ABAC_{sh}$ sentences formal description

	Model checking	**Satisfiability**
Proposed logic: ACL-DL	PTIME	PSPACE-complete
Propositional Logic	PTIME	NP
First-Order Logic	PSPACE-complete	Undecidable

It has been proven in Section 4.6 that ACL-DL model checking is PTime, which indicates that ACL-DL computing complexity is reasonable compared to First-Order Logic. ACL-DL matches the complexity of Propositional Logic, but regarding expressive power, Propositional Logic is not able to distinguish the level of truth as its validation mechanism is based on truth tables while ACL-DL is a modal logic.

Regarding the satisfiability problem, First-Order Logic is undecidable as it is a type of infinite model, and only under some fragments is it decidable. For example, ACL-DL can be translated to be FOL2 as a fragment of First-Order Logic with two variables. ACL-DL is formally proven to be PSPACE-hardness, as discussed in Section 4.6.

4.8 Summary and Discussion

Modal logic is appropriate for use in the access control field, as elaborated in Table 2.3 and investigated in Section 4.2. This chapter presents ACL-DL which combines access control logic features with policy rule specifications based on deontic logic.

Several studies on access-control logic have been proposed, mostly related to policy representation, enforcement, and proof-theory. The literature contains limited work that explores decidability, Kripke semantics, and the relation between the logics. The proposed ACL-DL logic in this chapter is able to represent policy rules through combining Access Control Logic with Deontic Logic. ACL-DL is decidable as it is a type of modal logic. Furthermore, ACL-DL has a defined Kripke semantic, in addition to the relations between the logics via ***says*** operator.

Chapter 5

$ABAC_{sh}$ Formal Specification and Logical reasoning via ACL-DL logic

5.1 Introduction

The logical approach is more appropriate for access control specification, verification and reasoning than calculus-based approaches as it allows easy detection of patterns of correct or incorrect reasoning [16, 47]. As clarified by [97] in Figure 2.4, access control modal must be formally specified in order to facilitate the access-control mechanism implementation phase.

This chapter employs the proposed ACL-DL in specifying and reasoning about the enhanced attribute-based access control, $ABAC_{sh}$. Therefore, it begins with an identification of $ABAC_{sh}$ properties, then proceeds with formal specification and logical reasoning in Sections 5.3 and 5.5, respectively.

5.2 $ABAC_{sh}$ properties

The main stages of $ABAC_{sh}$ model are described as followed and illustrated in Figure 5.1.

- The access control state before access request is triggered:
  - Define the entities set which is involved in the ABAC process. The entities are the subject, the object, the environment, and the system context
  - An Attributes Assignment function creates the attribute pairs (name:value). It assigns a set of attributes for each entity that reflect its character-

istics. The context attribute values are updated based on the current context state.

- Policy Creation function. This creates ABAC policy rules that reflect what sets of attributes access an action. Policy rules utilise an access capability format to create access rule entries.
- SoD rules creation. This creates rules to forbid a collection of attributes from performing a specific operation on an object.

- The access control state after access request is triggered:
  - Att():function to return attributes values.
  - SoD():function to check if SoD rules exist for the current request.
  - Policy(): function to check if permitted rules exist for the current request.
  - After rules checking, if the conclusion indicates that the access-request is permitted, then access is granted. Otherwise, access is denied.

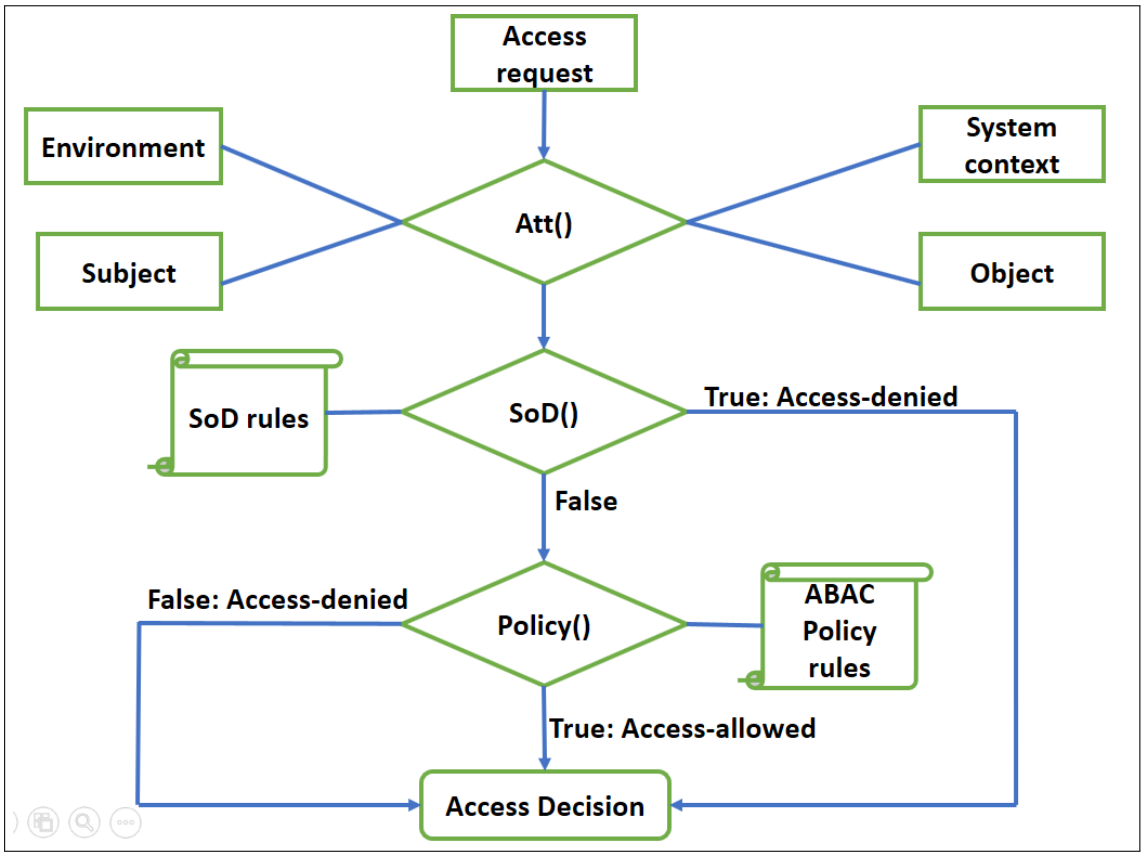

Figure 5.1: $ABAC_{sh}$ model flow-chart

The interpretation of the Kripke-structure model (Section 4.5) in the context of $ABAC_{sh}$ allows the definition of what it means for a formula to be true at a given

state in a structure. The $ABAC_{sh}$ formal model is in the form $\mathcal{M} = (\mathcal{W}, \mathcal{R}_1, \mathcal{R}_2, \mathcal{V})$ and described in the following sub-sections.

5.3 Formal Specification

ACL-DL logic is used to formally specify $ABAC_{sh}$ properties. Access control model formalisation requires the identification of a set of entities which will be active within the system and a set of actions. If the set of entities are denoted as E= { $enitiy_1$, ..., $entitiy_n$} and the set of actions A={ a_1, ..., a_n} . The decision for each action $\in A$ in relation to the set of entities in our $ABAC_{sh}$ model is either to be permitted or to be forbidden. The authorisation policy is expressed based on the ACL-DL logic. The authorisation process takes as an input the attributes of the subject, the object, the environment, and the system context to enforce the access-control rules.

5.3.1 Primitive Proposition

Primitive proposition indicates the basic facts which may hold in a system. In $ABAC_{sh}$, the action is represented as the primitive proposition. An action consists of an operation on an object **(opr, obj)**. Each action in the system must have at least one policy rule that controls its access permissions. A specific permission of an action is called a capability, which is considered a type of formula. For example, it is permissible for a Virtual Machine (VM) to be restarted. The primitive proposition is an action of restarting VM and denoted as **p**, where the capability represents the privilege of this action as to be permitted and formally specified using a deontic operator as $\boldsymbol{PE}(\mathbf{p})$.

5.3.2 $ABAC_{sh}$ possible worlds (states) $\mathcal{W}$

$\mathcal{W}$ is a non-empty set of $ABAC_{sh}$ states (or worlds). The world is described in terms of a non-empty set ϕ of propositional constants. Each state reflects one of the access control stages. Each stage indicates the system functionality and the possible states that involve knowledge and rules. The $ABAC_{sh}$ possible world elements are defined in Figure 5.2. The initial state involves defining the basic system elements such as the set of subjects, objects, and rules. Access control modelling starts when an access request is triggered; therefore, the first state is an access-request (w_1). The next state is related to an attributes assignment function Att(), where the attributes-gathering state takes the access-request knowledge from state (w_2) to return the required attribute values. After that, the state

quoting (w_3) uses the attribute sets to quote the access request on behalf of the subject, so the access-request sentence will be re-formulated. Then, the (w_3) state will feed (w_4) with the re-formulated access request to perform a SoD check. If the SoD() returns true, then there is a restriction on this action. Hence access is denied, and the access control process ends in the state (w_6) (access decision). Otherwise, if there are no restrictions on this access request, the (w_4) state will feed the (w_5) state with the re-formulated access-request to check if there is a policy rule to allow this action. If the access request has a rule that indicates that the requested operation on an object is necessary, then the process ends in the state (w_6), and the access decision will be allowed; otherwise, access is denied.

5.3.3 Binary Relation $\mathcal{R}$

ACL-DL has a binary relation of type serial between $\mathcal{W}$ elements. There is a relation between the states where one state is possibly getting information from the other state. Therefore, state w_2 must consider state w_1 in order for w_2 to make a decision. $\mathcal{R}(w_1,w_2)$ is known as an accessibility relation, as it defines the relation under which w_2 is a state accessible from the state w_1 . $\mathcal{R}(w_1, w_2)$ is also called a possibility relation, as it considers what states are required in a given state. Each state feeds the next state with knowledge in order to proceed, until the last state is reached, where an access decision is made, as explained in Section 5.3.2.

The initial state w_0 is not presented in the diagram as it holds the initial state of the system that initialises the basic system sets, such as the set of subjects, objects, and attributes. Figure 5.2 shows the serial relation of the world elements. There exist two relations, one for each modality. The ***says*** has relation $\mathcal{R}_1$ between the states that are responsible for preparing the access request to meet the $ABAC_{sh}$ properties by allowing the attributes to act on behalf of the subject that issues the access request. The obligatory (***OB***) relation $\mathcal{R}_2$ between the states is responsible for finding the access-decision between the rule sets. Therefore; $(w_1, w_2) \in \mathcal{R}_1$ from the access control point of view, represent a relation between states that are involved in the access-request formulation. While $(w_4, w_5) \in \mathcal{R}_2$ from the rules point of view, represent the relation between the states that are involved in access-decision determination. Only the successors (followers) of the current state can be inspected by the logic operators. The successors are states that are accessible by the current state through one $\mathcal{R}$-step. For example, if the current state is a w_4-SoD-check, then moving one $\mathcal{R}$-step reaches the successor state w_5:Policy_check, which is inspected via ***OB*** operator.

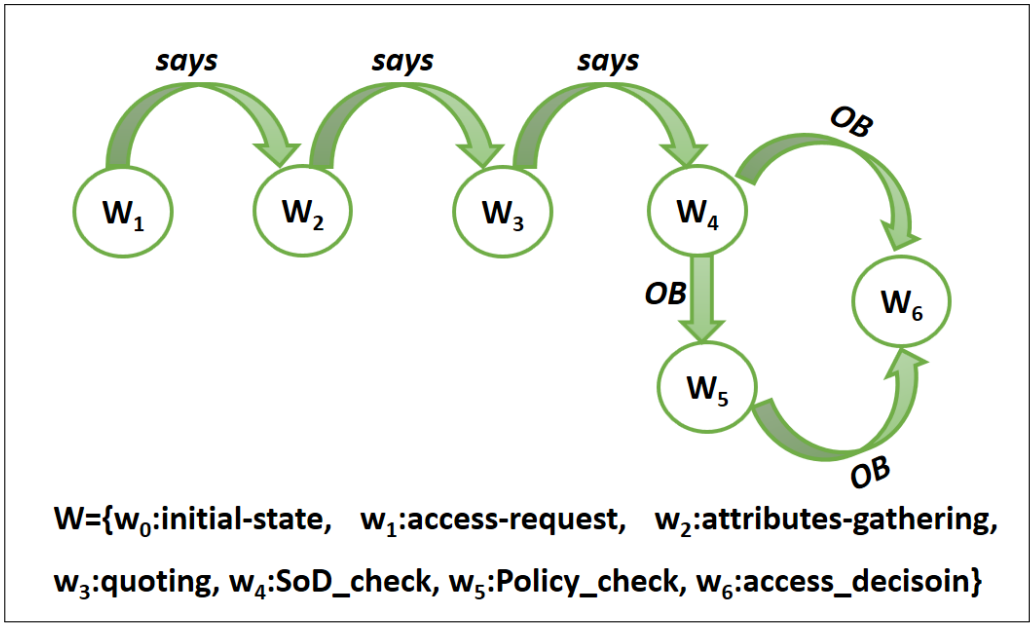

Figure 5.2: ACL-DL Serial relation for $ABAC_{sh}$ model

5.3.4 Mapping Function $\mathcal{V}$ and Rules Creation

$\mathcal{V}$ is a valuation function that captures the case in which the access request is allowed in a specific state of the system structure. Valuation is a critical function in access-logic as it associates each system statement (primitive proposition) which belongs to the system of well-formed formulas ϕ, a set of states in $\mathcal{W}$, therefore, $\mathcal{V}$ interprets at which state a primitive proposition is true. Unlike standard state notation, the state in a Kripke model is not completely defined using a valuation function. The SoD() and the Policy() are valuation functions where they give a true value to the primitive proposition statements at a specific system state. The SoD() indicates which access-request formula is true at state w_4:SoD_check, so that the next state will be w_6:access-decision, where the decision will be access denied due to the SoD (SoD rules). Otherwise, w_4:SoD_check state will have a relation with w_5:Policy_check state, if the access request entry has no constraint. The Policy() indicates which access request formula is true at the state w_5:Policy_check, where the policy rule indicates which access is allowed and which is denied. Rules creation in the proposed $ABAC_{sh}$ is in the capability format and is specified using deontic operators. The focus of our rules is on the object not on the subject. Therefore, it is more appropriate to use a capability mechanism instead of other mechanisms such as an access control list (ACL) [94, 47]. Deontic operators facilitate rule expression as they are able to describe various legal rules such as permissible($\boldsymbol{PE}$), obligatory($\boldsymbol{OB}$), optional($\boldsymbol{OP}$), impermissible ($\boldsymbol{IM}$), omissible($\boldsymbol{OM}$). Another wording for impermissible is forbidden ($\boldsymbol{FB}$). Therefore; the well-formed formulas in our proposed ACL-DL are able to formally present rule expressions.

Table 5.1: $ABAC_{sh}$ sentences formal description

No	System State	$ABAC_{sh}$ sentences	ACL-DL formal representation
1	w_0	Define finite sets of entities	Subject set={s1, s2, .sn}
2	w_0	Create primitive propositions finite set that reflect the system operations on objects	p=(opr→obj)
3	w_1	Access-request where a subject make a statement to perform an action	S says (opr →obj)
4	w_2	Attributes gathering function Att()	Att()=[name:value]
5	w_3	The access-request is reformulated as attributes quoting the subject	$S\|(Att(s)\&Att(o)\ \&Att(e)\&Att(c))$ says $(opr \rightarrow obj)$
6	w_4	Forbidden operation on an object within the current set of attributes based on SoD rules	OB(¬Access-request) FB(Access-request)
7	w_5	Authorized operation on an object within the current set of attributes based on ABAC policy rules	OB(Access-request)
8	w_6	Granted access request or denied	PE(Access-decision), FB(Access-decision)

The main difference between the SoD rules and the policy rules is that SoD represents constraints on specified objects, while each object must have at least one capability in the policy rules. Therefore, it is more likely that SoD-rules reflect forbidden actions, while policy-rules reflect what ought to be performed on an object.

5.3.5 $ABAC_{sh}$ Sentences Formal Description

Each row in Table 5.1 gives an example of $ABAC_{sh}$ sentences written in ACL-DL formal description. A collection of sets must be defined in the initial state w_0. Row no.1 gives an example of defining a finite set of the entity subject. The proposed model requires six sets: the subjects, the objects, the subject-attributes, the object-attributes, the environmental-attributes, and the system context-attributes. The $ABAC_{sh}$ model has two types of principals, either the subject that issues the access-request, or a set of entities attributes quoting the subject. The primitive proposition can be formally written as in row no.2. The access request state w_1 is described in row no.3, where a ***says*** operator is used to make a request.

5.4 Case Study

$ABAC_{sh}$ formal specification will be demonstrated through two case studies in this section.

The first example is shown in Figure 5.3 and is taken from [44]. This example illustrates an access permissions for a stuff named John. The formal representation of the ABAC policy rules Example 1 is listed in 5.2.

The second example from [73] is shown in Figure 5.4, and the formal specification of the ABAC policy using ACL-DL logic is illustrated in Table 5.3.

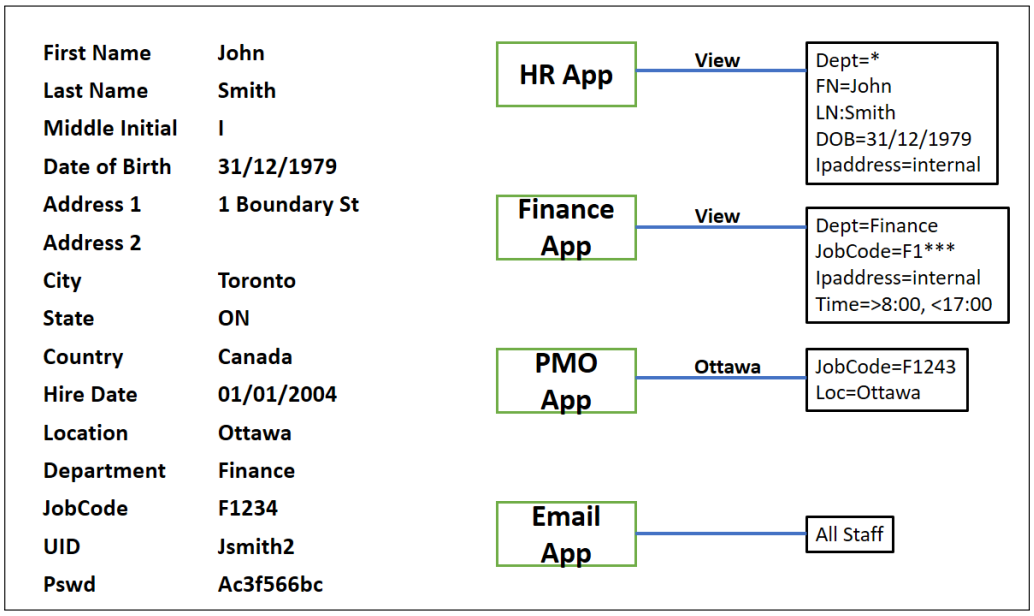

Figure 5.3: ABAC policy Example 1

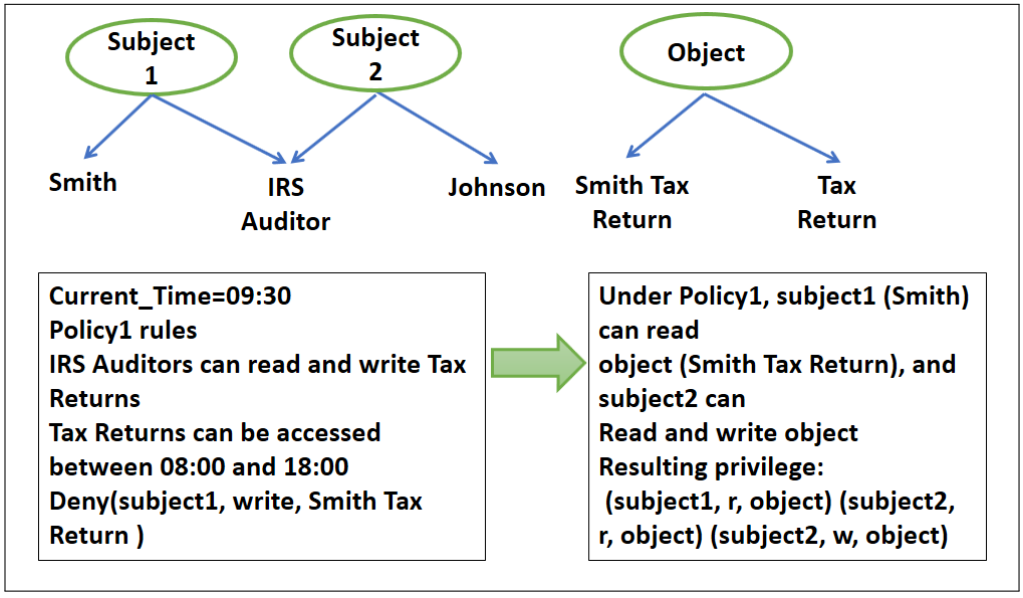

Figure 5.4: ABAC policy Example 2

Table 5.2: formal description for Example 1

Entities
Sub=John, obj1=HR App, obj2=Finance App, obj3=PMO App, obj4=Email App, Context= Network address and time, Environment=the organization has different departments
Policy Set
OB ([Att(sub)=(FN:John,LN:Smith,DOB:31/12/9797)]⋀ [Att(c)=IP:ipinternal] says (view →obj1))
OB([Att(e)=Dept:Finance] ⋀ [Att(s)=JobCode:F1***] ⋀ [Att(c)=NW:ipinternl, time:8:00 ⋁¡17:00] (says (view →obj2))
OB([Att(s)=JobCode:F1234] ⋀ [Att(e)=loc:Ottawa] (says (access project:Ottawa) →obj3))
OB([Att(e)=Dept:*] says (access →obj4))
Privileges set for John
PE(sub says (view →obj1))
PE(sub says (view →obj2))
PE(sub says (access project:Ottawa) →obj3))
PE(sub says (access→obj4))

Table 5.3: formal description for Example 2

Attributes
Att(sub1)= (Smith & IRS Auditor), Att(shb2)=(IRS Auditor & Johnson), Att(c)=(time:09:30), Att(obj)=(Smith Tax Return, Tax, Return)
Quoting
sub1|Att(sub1), sub2|Att(sub2), obj| Att(obj), context| Att(c)
SoD rules
OB ¬ ([Att(sub)=Smith⋀IRS Auditor] ⋀ [Att(obj)= Smith Tax Return]) says (write→obj))
Policy rules
OB([Att(sub)= IRS Auditor] ⋀ [Att(obj)=Tax Ruturn] ⋀ [Att(c)=time08:00 ⋁time¡18:00] says ((read→ obj) ⋁ (write→obj))
Privilege set
PE(sub1 says (read→obj))
PE(sub2 says ((read→obj) ⋁(write→obj))
FB (sub1 says (write→obj))

Deny exception in Example 2 can be described via SoD rules to reflect the constraint of the operation-object. Subject 1 attributes are restricted to perform the writing operation. Therefore, the resulted privileges allow Subject 1 to read an object. Subsequently, there are no restrictions on Subject 2; it can read and write on the objects, as both subjects have the attribute of IRS Auditor.

As indicated by the examples, there are several advantages to implementing an ABAC model policy over other access control models, as ABACs permissions are not based on user identity. ABAC reduces the risk profile as the access rights given are based on staff attributes. When the attributes are changed, the access

permission is dropped automatically without a need for a manual delete on the access permission list. There is also a lower cost for updating the access control rights.

5.5 Logical reasoning

The knowledge and understanding of this section are based on several investigations in the area, hence; the main references are [69, 114, 102]

5.5.1 Overview

Traditional access control reasoning mechanisms are based on a simple search on a table or a list to find the access matrix. However, there is demand for a rigorous reasoning mechanism due to the evolving complexity of access control systems. Therefore, access control reasoning approaches based on formal logic are adequate for access control systems, as illustrated in the related work in Chapter 2.

There are two steps to deploying logic in access control; firstly to define the logic which describes the access control model, and secondly, to define the reasoning mechanism [85]. Logic was defined to express ABAC_{sh} in Section 5.3. This section will demonstrate the proposed reasoning mechanism based on the inference which is used to verify the logical consequence of ABAC policy.

The category of reasoning followed in this research is called closed world reasoning, as all positive information is specified and it is concluded that any positive fact that is not specified or cannot be inferred will be false. Therefore, a complete knowledge has been assumed by specifying allowable access through a rule creation method as interpreted via a Kripke-structure. In contrast, there is an open world of reasoning used by classical logic such as First Order Logic [56].

Logical reasoning can be defined as a process followed to reach a new conclusion from a given premise [114]. Reasoning using the Kripke model is a semantic level. However, there is a requirement to add an extra inference rule to be able to reason the RBAC model [117]. The reasoning via inference rule is needed if the formal logic is not rich enough to express the policy specification or to capture certain system relations that cannot be described by the Kripke structure [85].

In this research, ACL-DL logic axioms and rules are sufficient to allow the use of the formulas to reason the ABAC_{sh} model.

5.5.2 Reasoning Methodology

In ABAC models, the access request will be granted if the collection of the attribute sets is permissible to perform an operation on an object. In the proposed reasoning

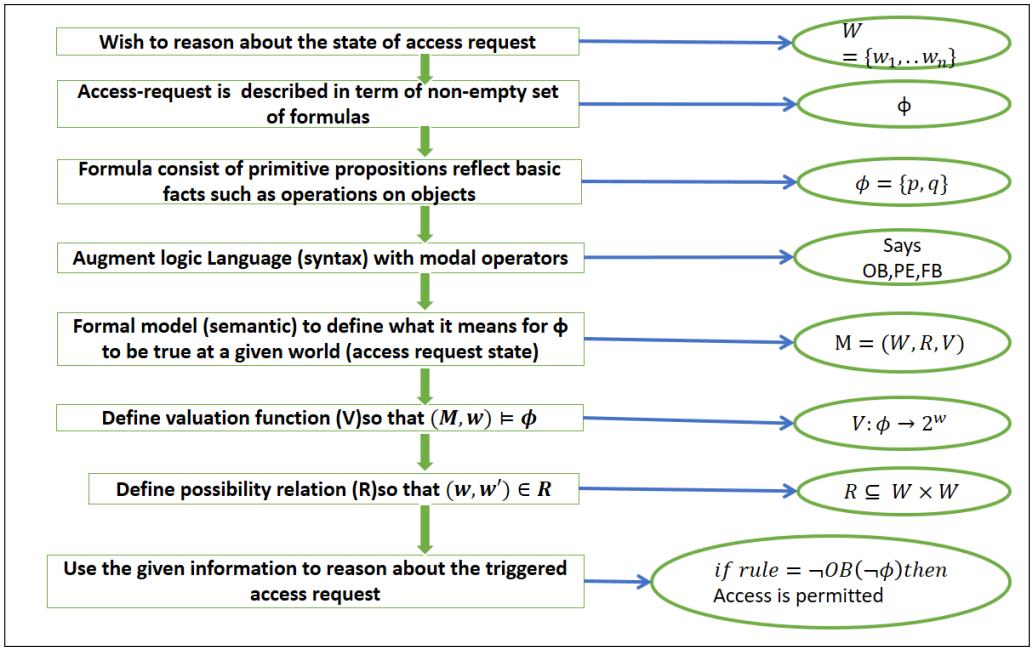

Figure 5.5: preparations for reasoning process

process, the logical consequences are based on the model-theory relations among the system defined the world, as explored in Section 5.3 and summarised in Figure 5.5.

In an ABAC model scenario, the information required for an access-request decision is not available beforehand. Hence, the obtainable information can provide a clue to the reasoning process. The available information is expressed in logical sentences ϕ, each of which gives a specific fact which may not represent everything about the world (access request state). These sentences can be combined into a logical theory, which in this research is a model theory.

Kripke structure is used as a model theory, where the proposed access control model is described in terms of states that are connected via relations, and truth values are given through a valuation function. This representation of access control model is used to draw a conclusion, where a set of true sentences (premises) entails a conclusion only if every state that satisfies the premises also satisfies the conclusion. However, it is not practical to check all possible worlds to find the logical entailment. Therefore, logical reasoning provides a tool in such scenario.

There are two types of logical reasoning: monotonic, and non-monotonic. The main difference between them is that monotonic reasoning follows a deductive inference approach where the information should be complete before the reasoning process starts, as is the case in First-Order Logic. In contrast, non-monotonic reasoning follows a defeasible inference approach where reasoning will be based on the available information. Therefore, if the information changes, the conclusion will also change. In this research, a non-monotonic reasoning approach is more

suitable as it suits the modal logic and artificial intelligence characteristics [23, 15].

5.5.3 Reasoning about $ABAC_{sh}$

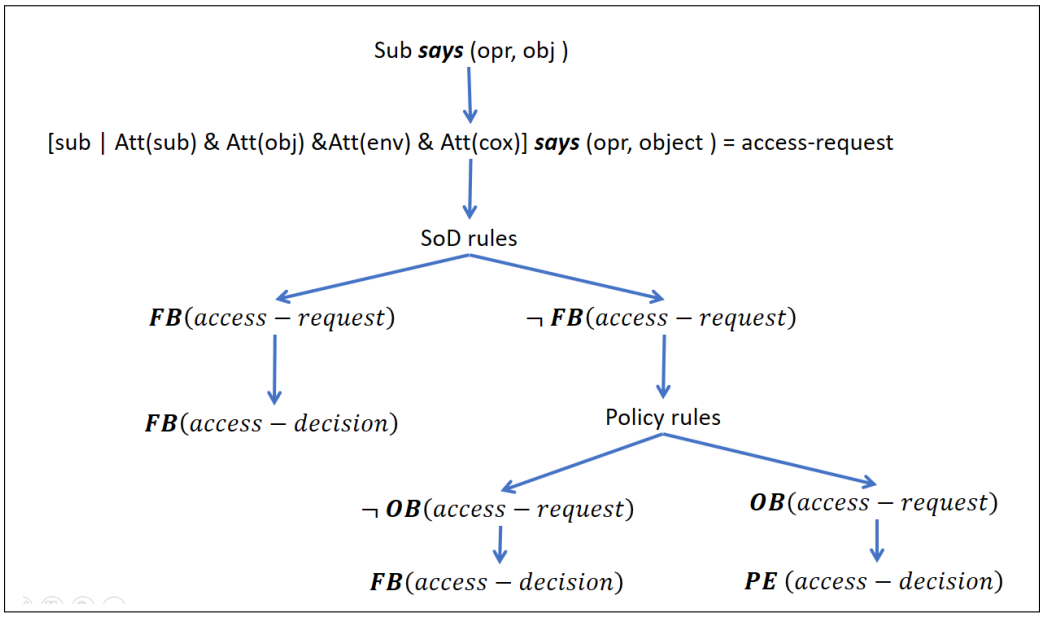

Figure 5.6: Reasoning process

The inference mechanism of the proposed $ABAC_{sh}$ model framework is illustrated in Chapter 6. If the inference mechanism can reach a conclusion on the ***PE***(Access-request), then the request will be granted, otherwise it will be denied. The inference mechanism is intelligent agent-based where the decision-making process is computed.

The applied reasoning method is based on a non-monotonic approach. Therefore, the access privilege decision is based on the available information from policy-rules and SoD-rules. Whenever rules are updated, the access privilege will also be automatically updated.

The access control argument consists of the set of sentences. Section 5.3.5 described $ABAC_{sh}$ formal sentences. The set of sentences that provide evidence called premises where the conclusion is the sentence that is acquired by the end of the reasoning process. In an $ABAC_{sh}$ model, there are three situations of premises and two possible conclusions. The first case is when there is a SoD rule indicating that access is denied. The second case is when there is an obligatory entry in the policy rules indicating that access is allowed. The last case is when there is no entry in the SoD or policy rules that fulfils the requested access, hence access is denied.

The reasoning process can be visualised as in Figure 5.6. The access request is only granted if the inference engine can make a conclusion that ***PE***(access-

decision). Such a decision is based on a series of access-control capability entries in SoD rules and policy rules.

5.6 Summary and Discussion

Formal logic is appropriate for use in specifying and reasoning about $ABAC_{sh}$ as elaborated in this chapter. The introduced formal specification (5.3) and logical reasoning (5.5) showed an accepted presentation for $ABAC_{sh}$ model relations and states which leads to better understanding of how this access control is working. This facilitates the next stages which are the design phase (Chapter 6) and implementation phase (Chapter 7).

ACL-DL which is a type of modal logic shows an expressive representation for $ABAC_{sh}$. Employing logic in access control model specifications, avoid the ambiguity and allow a formal representation. Moreover, it helps administrators to automatically derive the consequences of their policies.

Chapter 6

An Intelligent Framework for $ABAC_{sh}$

6.1 Introduction

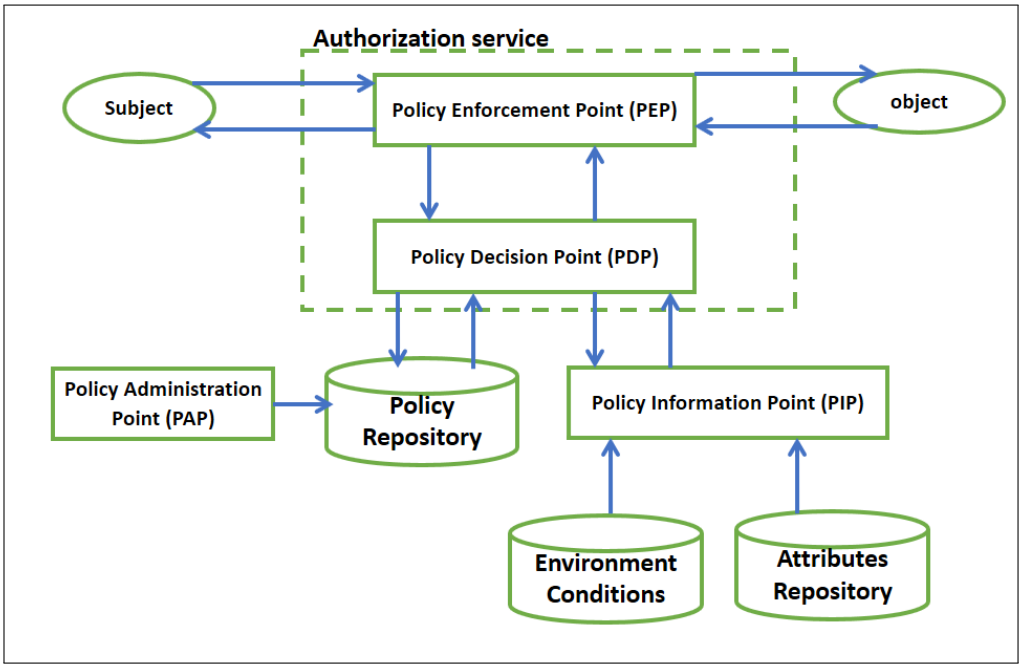

Figure 6.1: ABAC mechanism function points based on XACML framework

This chapter will present the design of the proposed $ABAC_{sh}$ model framework, taking into consideration the essential ABAC elements and a knowledge-based agent architecture. ABAC's underlying mechanism function points are illustrated in Figure 6.1, extracted from an XACML (eXtensible Access Control Markup Language) framework [98, 157] . The main four points are PEP, PDP, PIP and PAP: the Policy Enforcement Point (PEP), where the access decision is enforced; the Policy Decision Point (PDP), where the access control is processed to produce

a decision; the Policy Information Point (PIP), where the information related to access control processing is supplied to the PDP, and the Policy Administration Point (PAP), where the system administrator feeds the system with the access control policy rules. The artificial intelligence categories involved are agent architecture and logical reasoning. The aim of proposing an intelligent framework for $ABAC_{sh}$ is to design a dynamic attribute-based access control that can think and act rationally. The following sections will illustrate how artificial intelligence is utilised to meet this aim.

6.2 AI scope for the Proposed Framework

According to [69, 149, 182], artificial intelligence systems are designed to think and act. They can be categorised into four types based on the intention of the system: Thinking Humanly, Acting Humanly, Thinking Rationally and Acting Rationally.

The category of Thinking Rationally leads to an evolved need for the logic field in artificial intelligence. Involving logic in an intelligent system faces two substantial obstacles. The first one is the difficulty of presenting informal-knowledge using a formal logical notation though the certainty level is less than 100%. The second is that solving problems theoretically is different from solving them practically when the machine capacity is taken into consideration.

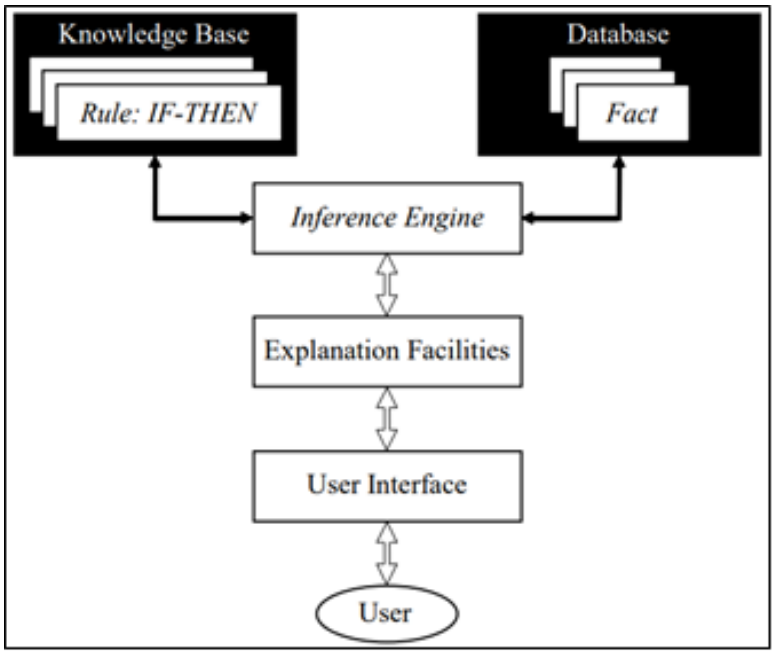

Figure 6.2: Basic structure of a rule-based expert system (Negnevitsky 2011)

The category of Acting Rationally initiates the development of a computer agent. Prior to computer science, the term agent was used in different fields.

Therefore, there are various definitions of agent. However, it can be defined as an entity that acts within an environment by sensing its surroundings to update its knowledge and acts upon that to meet specific goals [199]. The agent function represents an abstract mathematical description, whereas the agent program represents an agent implementation within a physical system.

Problem-solving through an intelligent agent involves four stages. Firstly, the agent formulates its goal. Secondly, it formulates the problem based on five steps: initial state, possible actions, transition model that describes what each action does, goal test and path cost. Thirdly, it searches for a solution by looking for a sequence of actions that leads to the goal. Fourthly, in the execution stage, the solution found is implemented. However, the problem-solving agent is inflexible as each possible state should be hard-coded. Therefore, the complexity of the search stage grows exponentially in relation to the number of states in addition to its inability to infer unobserved information. Therefore, there is a need for logic to reason about the possible states instead of hard-coded all predicted states.

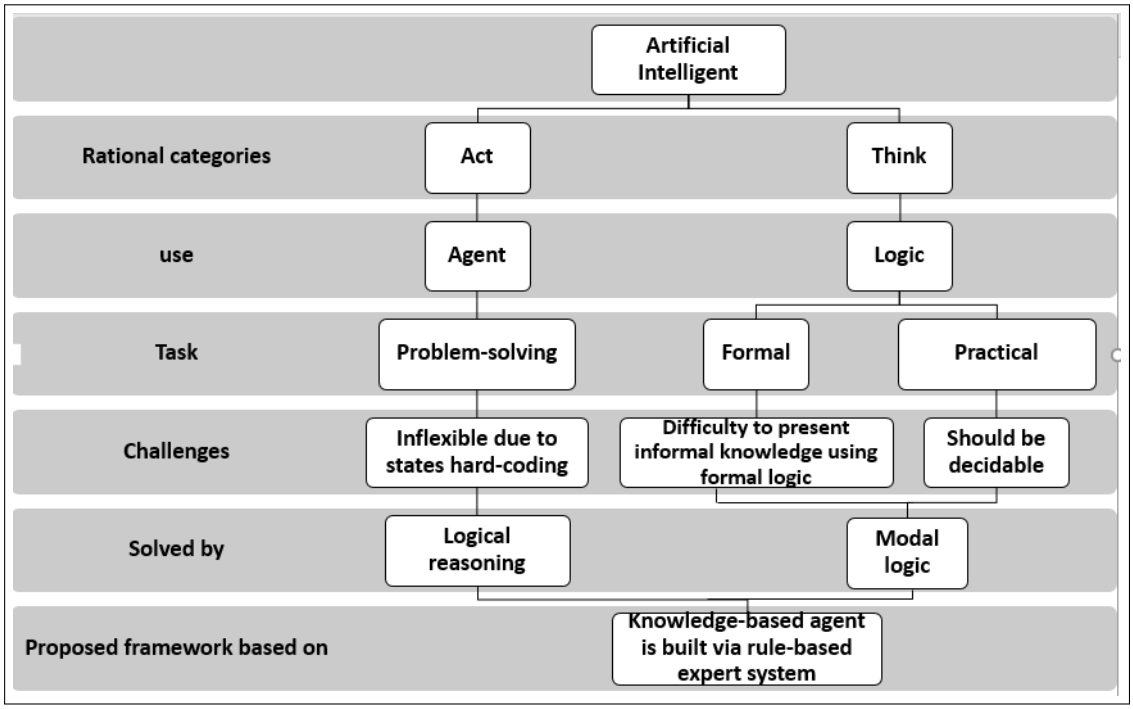

Figure 6.3: AI scope for the proposed framework

Knowledge-based reasoning is a step in overcoming problem-solving agent limitations. The logic provides a natural language for describing and reasoning about the system. The knowledge-based system is given facts about the external world, and it is asked queries about that world. The rule-based expert system is a popular method that is used to build knowledge-based systems. The rules are used to represent knowledge in the format of IF-THEN. The structure of a rule-based expert system is illustrated in Figure 6.2. The Inference engine is the reasoning

component whereby the system concludes by linking the rules given in the knowledge base with facts supplied from the database. The explanation facilities allow the user to interact with the expert system to get justification regarding the results produced by the inference engine.

Therefore the AI scope for the proposed intelligent-framework for $ABAC_{sh}$ is illustrated in 6.3. Modal logic is found to be the most appropriate logic to be used in AI as discussed by [140] and highlighted in Section 4.2 .

6.3 Logical-Based Agent Architecture

Intelligence security is a fertile approach, as most existing security paradigms suffer from reactive and fragmented approaches [115]. In a frequently changing infrastructure, deploying an agent-based mechanism will be an advantage [63]. Modal logic is a candidate that supports a logical approach in artificial intelligence systems [102]. The main component of a knowledge-agent is a Knowledge-Base (KB) that consists of a set of sentences expressed using formal logic, in addition to two generic functions that involve logical inference. The first function is known as TELL, and adds new sentences (facts) to the KB to provide it with the required information. The second function is known as ASK, and queries the known information from the KB to determine the next step. The process between TELL and ASK will end as soon as the desired action is selected. The interaction between these two generic functions is similar to the updating and querying in databases, as illustrated in Table 6.1. When an agent program is called upon, it performs two main actions. Firstly, it will TELL the KB what it perceives. Secondly, it ASKs the KB what action should be taken.

Table 6.1: A generic Knowledge-based agent function [182]

function KB-AGENT(percept) **returns** an action
static: KB, a knowledge base
t, a counter, initially 0, indicating time
TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
action ← ASK(KB, MAKE-ACTION-QUERY(t))
TELL(KB, MAKE-ACTION-SENTENCE(action,t))
t ←t + 1 **return** action

Therefore, agent-based architecture is suitable to represent an ABAC model. The logical agent, furthermore, will be appropriate for the proposed modal logic scheme. Table 6.2 demonstrates how knowledge-based agent architecture can represent an ABAC system. The logical agent can be designed to represent an access-request state through a process of inference to derive a new representation

Table 6.2: mapping knowledge-based agent with ABAC requirements

Components	agent architecture	ABAC requirements
knowledge base(KB)	Background sentences	Predefined sets of entities, attributes and policy rules
	To represent action(s)	The action is access-decision
inference system	Infer (i.e arrive to a conclusion via reasoning) hidden properties of the world to add new sentence to KB	New sentences are added each time an access-request is triggered which consist of a combination of attributes with the request operation
	Infer based on the predefined sentences and the new ones to conclude with appropriate actions	Reasoning based on the attributes set and the policy rule-sets to conclude with an appropriate action (allow or deny) the access-request

of the access-request state that can be used to deduce required actions. The proposed access-control logic agent will be founded on knowledge-based agents, as this type of agent is logic-based [182].

6.4 $ABAC_{sh}$ Conceptual Requirement

Based on the analysis and investigations addressed within Chapter 2 and Chapter 3, the critical requirements in designing an ABAC model are listed below.

Req.1 ABAC model definition requires to identify the configuration points. Each point should be formalised via the proper languages. The configuration point indicates the necessary configurations to be accomplished via the ABAC model processing for computing the access decision. These points are known as functional points. It is more convenient to minimise the number of configuration points as they affect the system's computational complexity.

Req.2 ABAC is identity-free. Therefore, identifications such as subject-id are not the main elements in access-decision processing.

Req.3 Avoid the creation of lists or groups in the design, as ABAC is intended to be fixable and able to cope with large enterprises.

Req.4 Context-attributes reflect the current system state, whereas environment-attributes reflect the fixed system characteristics.

Req.5 ABAC is a multi-factor decision. Therefore it enables fine-grained access control.

Req.6 There are no predefined privileges for subjects as the privileges are computed after an access request is triggered. Policy rules set in ABAC are specified based on attributes. As a result, the permissible operations will be defined upon access-request.

Req.7 The two basic functionalities in ABAC are attribute-assignment and rules-creation.

Req.8 Security principles such as Separation of Duty (SoD) must be enforced.

The enhanced attribute based access control ABACsh fulfils requirement Req.1 by employing one main configuration point that is ABAC agent. This agent takes as an input, the access request parameters which consist of the subject, the object, and the operation (s, o, opr). Then it returns the access decision that indicates if the subject is allowed to operate on the object or it is denied. Compared to ABACα, which has four configuration points, the policy configuration here is reduced to one, as the proliferation of policy configuration points can introduce difficulties in policy expression and comprehension [108]. For Req.2, in the Policy Decision Point (PDP), the decision-making process considers the subject attributes in addition to other attributes, instead of depending solely on the subject identity information. In Req.3, grouping is studied by HGABAC [192] to facilitate the addition of a hierarchy feature to ABAC. However, grouping and listing will impede the flexible nature of ABAC [52, 98]. Therefore, permissions grouping and listing are avoided in this $ABAC_{sh}$. The decision calculation is based on four sets of attributes: subject-attributes, object-attributes, environment-attributes, and system-context attributes, all of which are taken into consideration in the proposed design to meet requirement Req.4. System context attributes have a special sensor to obtain an up to date system state to meet requirement Req.5. The privilege decision is calculated based on the attributes relation defined in the policy-rules. Therefore; the privilege value is returned after the access-decision is triggered, which meets the requirement Req.6. There are two core functions of $ABAC_{sh}$. The first function takes place at the initial system stage, where the attribute pairs (name:value) are created for the defined access control system entities (subject, object, environment, and context). The second function is rules creation, which represents the SoD-rules and Policy-rules in the form of capability which indicates the access-rights. These two functions meet the ABAC requirement Req.7. An initial SoD is introduced in this design in the type of a DSoD. The elimination of policy-rule conflicts can be achieved by an object-operation oriented constraint. A formal presentation of the proposed SoD enforcement sentences is defined and will be flexible to manage the set of constraints and meet the system

requirements Req.8 since the administrator can modify the set of SoD sentences. The proposed SoD will be enforced after the access-request is triggered where an action is forbidden based on a collection of attributes.

6.5 The Proposed Framework

The proposed $ABAC_{sh}$ model framework focuses on three functional points in reference to XACML framework: PDP, PIP and PAP as illustrated in Figure 6.4. The framework flow is described below:

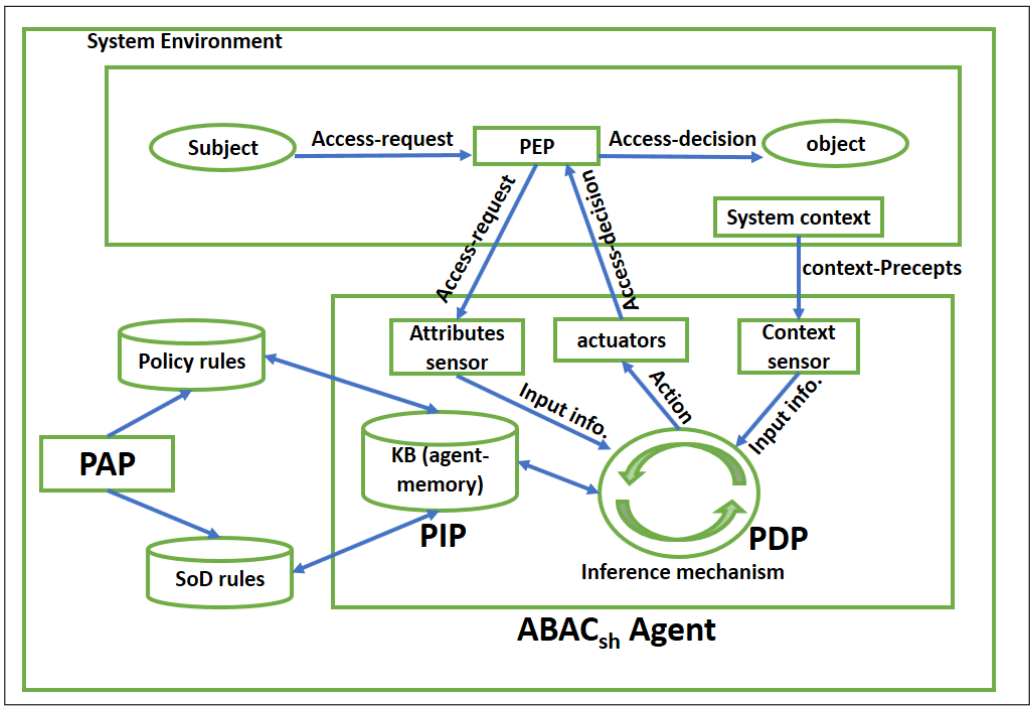

Figure 6.4: The proposed Intelligent Framework for $ABAC_{sh}$

- The main access-control entities, which are subject and object. The subject issues the access request. The object will provide the resources if the request is authorised.
- The Policy Enforcement Point (PEP) enforces the access decision. PEP can be a firewall as it depends on the implemented security appliance type. It takes the access request as an input and sends it to the PDP. It receives the access-decision from PDP and enforces it into the object.
- The ABAC agent is the logical access-control component. It is built based on knowledge-based agent architecture. It involves two main configuration points based on an XACML framework. The first is PIP, where the ABAC model collects the information needed in an access-decision process. The

second is PDP, where the information collected from PIP in addition to the access-rules provided by PAP are used to infer an access decision.

- The PIP involves information collection as follows:
  - A sensor that collects the attributes needed requested by attribute function Att()
  - A context-sensor that collects the current values of the system context-attributes
  - Information that reflects the domain-specific content is stored in KB data storage
- The PDP involves the core logical reasoning that takes place in an inference mechanism where the access decision is processed. The inference mechanism is the logical engine where a domain-independent algorithm is used to compute the access-decision and return an action value through the actuators to the PEP.
- The PAP involves rule creation by the system administrators. This point involves two types of rules, as follows:
  - Policy rules contain a predefined set of policies
  - SoD rules that reflect constraints to avoid policy-rule conflicts.

The main ABAC$_{\text{sh}}$ processing stages that are required to be formally described are attribute assignment, policy rules, SoD rules, and access requests, as illustrated in Figure 6.5. Based on this information, the inference mechanism can draw a conclusion as to whether the requested access is to be allowed or denied.

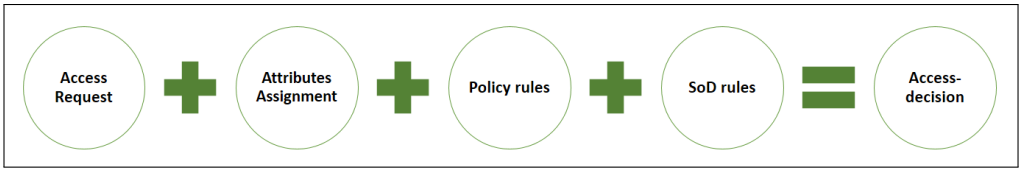

Figure 6.5: ABAC$_{\text{sh}}$ processing stages

6.6 Summary and Discussion

Artificial Intelligence mechanisms are recommended to be used in a dynamic and rule-based system such as ABAC in IaaS cloud. Two AI categories are utilised in

the proposed intelligent framework for $ABAC_{sh}$. These categories are Act Rationally and Think Rationally as illustrated in Figure 6.3 and discussed in Section 6.2. The Act Rationally challenges are solved by using formal logic while the Think Rationally challenges are solved by using modal logic.

The framework is designed based on knowledge-agent and it employee rule-based expert system method. This intelligent system is not based on machine learning which will have a percentage of correct answers. This system is based on the available rules, therefore it is not a type of uncertain approch. The system must guarantee an access decision. In reference to the framework functional points in Section 6.5, an enforcement architecture aims to implement them through IaaS platform as will be demonstrated in Section 7.3.1, whereas, a prototype implementation of $ABAC_{sh}$ will be discussed in Section 7.3.2.

The purpose of this $ABAC_{sh}$ framework is to prove that AI architecture can contribute in supporting a dynamic access control which meets IaaS characteristics. In regards to guaranteeing behaviour, the followed mechanism in this chapter is based on knowledge available. If there is a shortage in knowledge, the access decision will be denied. There are other AI categories related to uncertain knowledge such as probabilistic reasoning, However, uncertain reasoning is out of this research scope.

Chapter 7

$ABAC_{sh}$ Implementation for IaaS Via OpenStack

7.1 Introduction

This chapter demonstrates the visibility of $ABAC_{sh}$ in IaaS cloud by introducing an enforcement architecture based on OpenStack. That is followed by a prototype implementation and performance evaluation that illustrates the advantages of the proposed $ABAC_{sh}$ extension over the existing access control model. The contributions of this chapter are as below.

- Designing enforcement architecture for $ABAC_{sh}$ that utilises telemeter service deployment to be used in feeding Policy Information Point (PIP) with attributes values.
- A prototype implementation of an extended nova access control model with an intelligent ABAC.
  - Extend the nova policy enforcement point (PEP) to communicate with an external policy engine
  - The proposed external policy-engine works as policy decision point (PDP)
  - The introduced PDP follows $ABAC_{sh}$ by
    - Utilising the attributes in access decision-making process.
    - Involving forward-chaining algorithm that works as logical reasoning for access decision processing.
- Three experiments are studied to compare and contrast the extended $ABAC_{sh}$ with the default nova-OpenStack access control model.

- The Quality of Service (QoS) measurement is discussed based on response time as a performance metric.

This chapter has four main parts. The first part introduces an overview of the primer concepts which are considered as a base for building these chapter contributions. The preliminaries are an OpenStack overview, OpenStack Access Control Model (OSAC), Policy Engine and forward-reasoning algorithm. The second part demonstrates $ABAC_{sh}$ enforcement architecture for OpenStack and the prototype implementation. The third part illustrates the performance evaluation with the results and discussion. The fourth part presents comparative study of ABAC models implementation in OpenStack. This analysis involves $ABAC_{sh}$ and two ABAC from the literature which are ABACα and HGABAC.

7.2 Preliminaries

7.2.1 Open source IaaS-Cloud Test-bed

A study of the implementation tools in Cloud computing environment was studied by [185]. The tools were classified as below:

- Mathematical modelling: Suitable for finding optimal values and predicting behaviour
- Simulations: Does not represent an actual Cloud such as CloudSim, GreenCloud, iCanCloud and teach cloud.
- Testbed: Hardware platform which can be a commercial-based such as Amazon EC2, Google App, Windows Azura, or a research-based such as OpenCloud, OpenCirrus. Software platform such as OpenNebula, Nimbus, OpenStack.

In order to experiment and test the access control mechanism of IaaS, there is a need to investigate the available testbeds to find a platform that allows source-code modifications. Therefore a brief study in the available open source test-beds has been made. The results of the literature are summarized in Table 7.1.

Figure 7.1 illustrates the numbers of hits of popular Cloud tools from leading academic libraries. Amazon EC2 is not open source, but it has been used by many academic authors to validate their work, as its fees are cheaper than IBM Cloud. Eucalyptus has an active committee and it is used by Sony, Puma, NASA, Trend Micro and other companies to build their private Clouds, but it does not support the LVMM process [213].

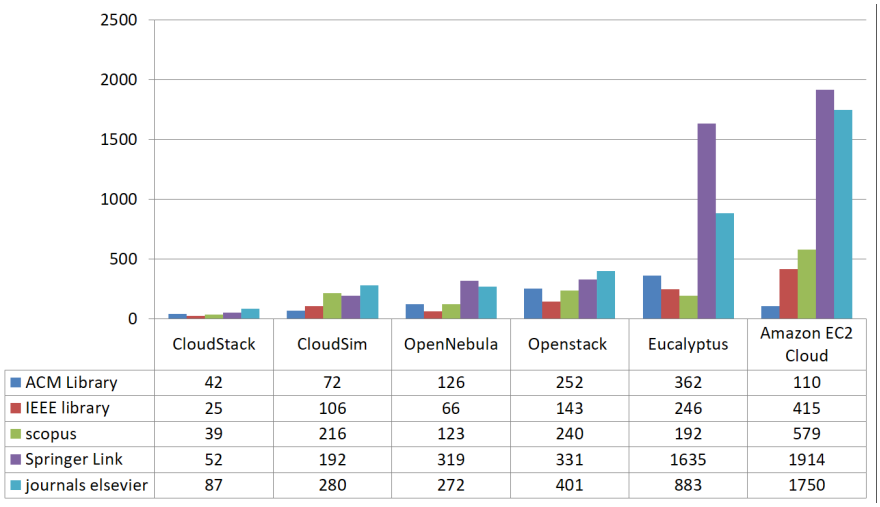

Figure 7.1: Number of hits in variety of academic search engines related to IaaS platforms

Table 7.1: OpenStack core components

Ref.	testbed options	Aim of the Study	Observations
[171]	OpenNebula, Eucalyptus, OpenStack	A study to find out which to use in their private cloud	OpenNebula easier in deployment where OpenStack is more complex but has and active community
[151]	OpenNebula, Eucalyptus, OpenStack	To evaluate operation of web-hosting service and Cloud Abstraction Layer (CAL)	OpenNebula take more time for deployment the the rest while Openstack was the faster.
[12]	OpenNebula, Eucalyptus, Nimbus	Comparing middleware functionalities	The Least one which support security standards is Eucalyptus. The least one which support scheduling is OpenNebula while it is the best in virtual management support
[225]	OpenNebula, Eucalyptus, OpenStack	deployment recommendations to different user demands and platform characteristics	Two of test-bed support similer VMM except that OpenNebula doesnt support HyperV and Xen Server

The CloudSim simulator has been explored to study its abilities to perform the initial experiments. However, it does not support security implementation as it requires JAVA extension classes in order to simulate security components. Moreover, CloudSim does not perform like an actual Cloud.Openstack has been

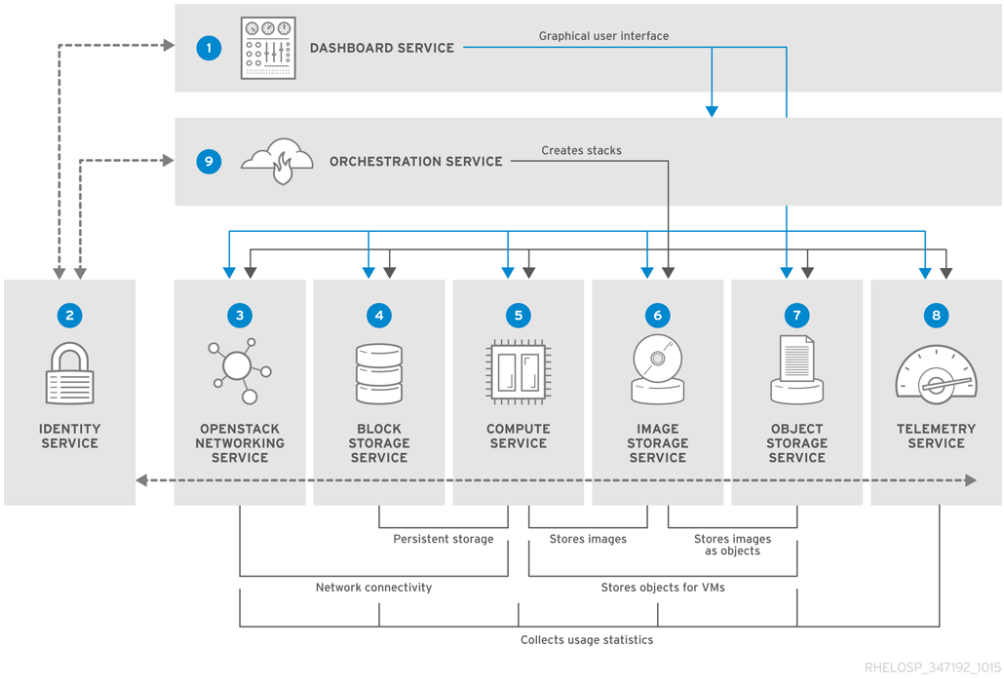

Figure 7.2: OpenStack Components [164]

selected to be the experiment platform for this PhD thesis as it support the required prototype implementation and has a very supportive community of both academic researchers and commercial bodies.

7.2.2 OpenStack Overview

A key component in building a virtualisation environment is its operation via the hypervisor. The hypervisor on its own cannot build IaaS. Therefore, a cloud-stack such as OpenStack, Cloud-Stack or OpenNebul is required. According to the current industry, OpenStack is likely to become a dominant cloud-stack [100]. OpenStack is an open-source cloud computing platform that offers an IaaS layer of service. OpenStack IaaS infrastructure supports agent communication. For example, network nodes in the OpenStack activate a DHCP agent to deploy a DHCP service [160]. OpenStack was selected to be the experimental platform for this research as it has a supportive and active community of both academic researchers and commercial bodies. Figure 7.2 illustrates the OpenStack components [164] which are known as OpenStack projects as well.

The core components are described in Table 7.2 [162] and are keystone, glance, nova and neutron. The remaining components are optional and are dashboard service (known as horizon), block storage service (known as cinder), object storage service (known as swift), telemetry service (known as ceilometer) and, orchestration service (known as heat).

Table 7.2: OpenStack core components

Component	Known as	Description
Identity Service	keystone	An integrated management single point that provides authentication, authorization, and a catalogue of services. It consists of a server, drivers and modules. The server is based on the RESTful interface. The drivers are back end services. The modules are middleware run in Openstack components to communicate with keystone during the authorization process. The modules use Python Web Server Gateway Interface.
Image Service	glance	A centralised registry service to store resources such as virtual machine (VM) images. The image is an operating system that can be installed in VM.
Compute Service	nova	A major component in IaaS system that manages and hosts VMs running in the hypervisor. Its main modules are implemented in Python.
Networking Service	neutron	Manage networking services for Openstack IaaS. Its common agents are Layer3 and DHCP.

Figure 7.3 demonstrates the main interactions between the OpenStack components. The main task for IaaS is to provide the virtual machine (VM) to users. Thus the VM is indicated in a diamond shape. The keystone provides an authentication service between the users and the OpenStack system and between the OpenStack components themselves. The nova boots the instance processing data and provision VM. The glance provides VM with the image. The neutron provides network connections for VMs.

7.2.3 OpenStack Access Control Model (OSAC)

OpenStack can deploy different access control models within its infrastructure [162]. For example, nova configuration files can be protected via several implementations such as centralised logging, policy file (policy.json) and MAC framework (Mandatory Access Control). The availability of access control models depends on the hypervisor vendor. The supported models up to the date of writing this thesis are Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role-Based Access Control (RBAC).

The Openstack access control model (OSAC) that enables both operators and users to access resources for specific services is a type of RBAC. A simplified OSAC is illustrated in Figure 7.4 [25]. The keystone [163] supports the notation of roles and groups. Each user should be associated with a group, and each

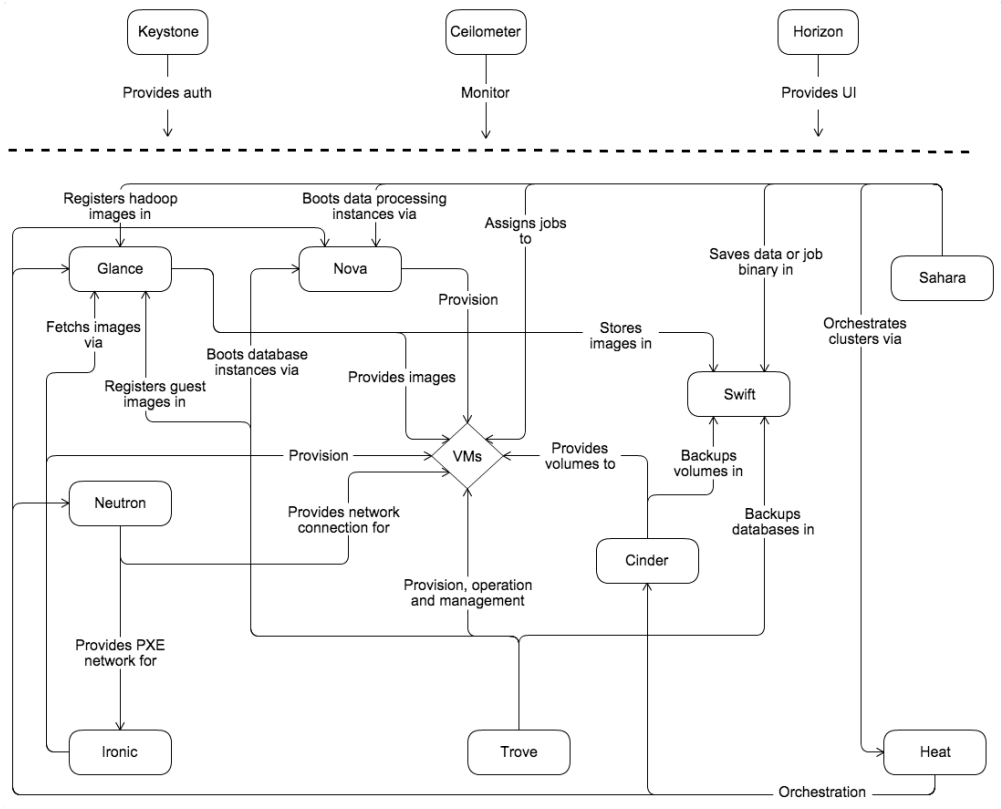

Figure 7.3: OpenStack Conceptual Model [161]

group has a list of roles. For a user to be granted access to a service, Openstack service takes into consideration his/her role, though as the first authorization step, the OpenStack PEP (Policy Enforcement Point) takes into consideration the policy rules associated with the resources before it checks the user role. Therefore, the policy enforcement middleware enables fine-grained access. Each Openstack service defines the access control policies rules for its resources in a specific policy file called policy.json.

[77] The OpenStack manages two types of authorization policy: operation based and resource based. The operation based specifies access criteria with specific attributes where policy.json is used, while the resource-based uses RBAC and it is available to network resources only up to the date of writing this thesis.

The OpenStack access control faces the followed challenges

- An empirical demonstration for Hacking nova through a fake user id [135]
- The policy engine is embedded within the code where it is recommended to be separated [169]. Therefore, congress policy framework is introduced in the OpenStack cloud for developers [168].
- Because of the admin role is set as global, a Bug 968696 is occurred. Therefore, there is a need to investigate the following to solve this problem [230]

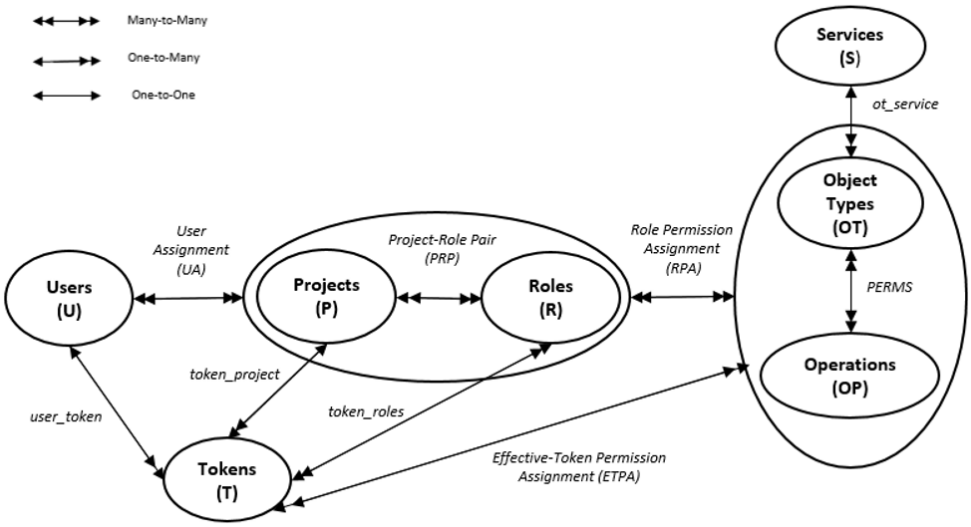

Figure 7.4: OpenStack Access Control (OSAC) [25]

- Merge Policy Header
- Unify Nova, Glance and Cinder
- Move towards a single File
- Break member up into smaller roles
- Remove isadmin from lines
- Fetch policy via Endpoint Policy Extension

7.2.4 Policy Engine

Policy engine in OpenStack is a type of authorization engine that return back a decision based on some policy rules that indicating if a specific operation is allowed or denied [230, 25, 162]. The default policy engine is maintained via Oslo policy, and the access request is issued via API communication. Oslo policy is completely separated from RBAC model [166]. The developer can view Oslo policy rules that are related to nova via the command "oslopolicy-policy-generator –namespace nova ". The list of rules verifies if the user credentials are matching to grant access to the requested resources. The user credentials are stored in the format of a token. The token holds information related to the token itself in addition to the user, the project, the domain, the role, the service and the endpoint. The policy rules are stored in JSON (JavaScript Object Notation) file format.

In policy.json, the access policy consists of two main parts "<target>" : "<rule>" [163]. The target is known as an action that indicates the API call for an operation such as "start an instance". The rules can be one of the following: allow all, deny all, a special check, a comparison of two values, Boolean expressions based on

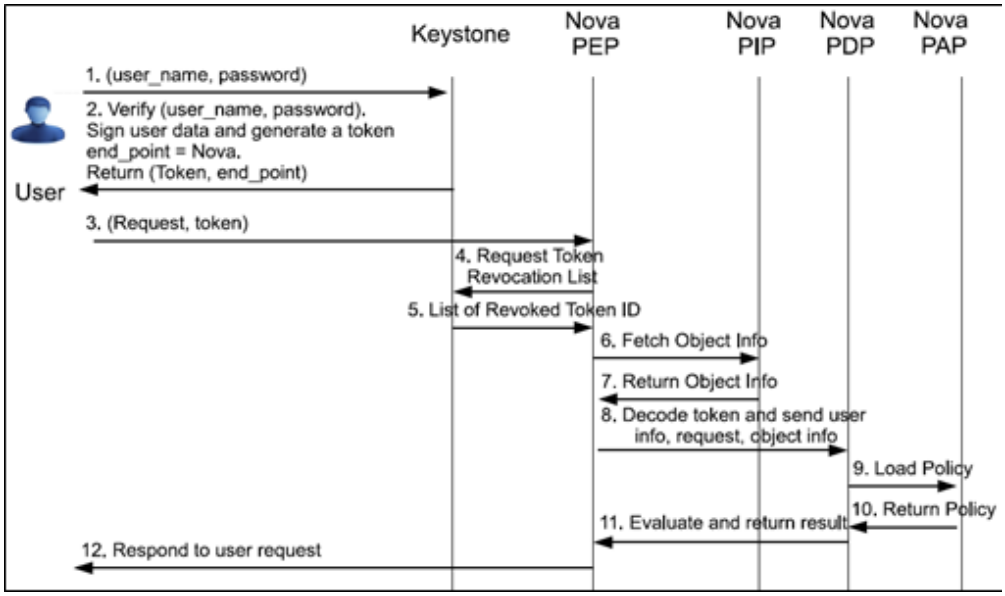

Figure 7.5: Nova authorization data-flow [109]

simpler rules. The special check gives the developers an opportunity to extend the OpenStack policy engine. The special check can indicate, a role that is extracted from token information, or a complex rule by defining an alias, or a target URL that delegates the check to an external policy engine.

7.2.5 Nova Authorisation Data-Flow

Each service in Openstack has its own access control configuration points which involve PEP, PIP, PDP and PAP. The information flow between nova access-control configuration points is demonstrated in Figure 7.5.

In the original Openstack architecture, `Nova_PEP` will send a token that contains the information of the access request to `Nova_PIP` to retrieve the object information. Then `Nova_PEP` sends the information of the subject, object and request to `Nova_PDP` in order for `Nova_PDP` request an access control policy from `Nova_PAP`. `Nova_PDP` evaluate the access request based on the policy and return the access decision to `Nova_PEP`.

7.2.6 Forward-Chaining Algorithm

In the search stage of the problem-solving agent (as explained in Section 6.3), there is a need to use a proper searching algorithm that meets the problem scenario and the input information. The search algorithms that are used in rule-based systems are backward chaining, forward-chaining and a mixture of both of them [88, 182]. Table 7.3 compares between the reasoning algorithms which are referred to as chaining in some literature.

Table 7.3: Comparing Forward-chaining with Backward-chaining

	Forward-chaining	Backward-chaining
Known as	Forward reasoning (Data driven)	Backward reasoning (Goal driven)
reasoning Start with	A Set of facts to reach a goal (or hypothesis)	A hypothesis (goal) to reach the facts behind it
When applicable	If the goal is unknown	If the set of goals are known

Table 7.4: Forward chaining algorithm

newFacts= False
For rule in rule-list:

- If all premises match fact-base:
  - For each fact in consequences:
    - if fact not in fact-base:
      - add fact to fact-base
      - newFacts = True

If newFacts: goto 1

Many researchers avoid the Logic Theory Machine, which is based on forward-reasoning due to the computation complexity. However, this complexity is due to the classical mathematical logic and is not due to the forward-reasoning concept [46]. Classical mathematical logic such as propositional logic and First-order logic. Therefore, the computational complexity of forward-chaining when it is used in nonclassical logic such as deontic logic will be decidable, and it will have an acceptable computation complexity. A simple algorithm for forward-chaining is illustrated in Table 7.4. Forward reasoning search iteration is based on facts and rules to find a conclusion.

7.3 The Proposed $ABAC_{sh}$ Implementation for OpenStack

7.3.1 Enforcement Architecture

The core characteristics of $ABAC_{sh}$ are to work as an intelligent agent that sense the attributes (the environment, the system context, the subject and the object) in order to search for an access decision using forward chaining (forward-reasoning). The set of attributes represent facts whereas the set of policy rules represent the rules.

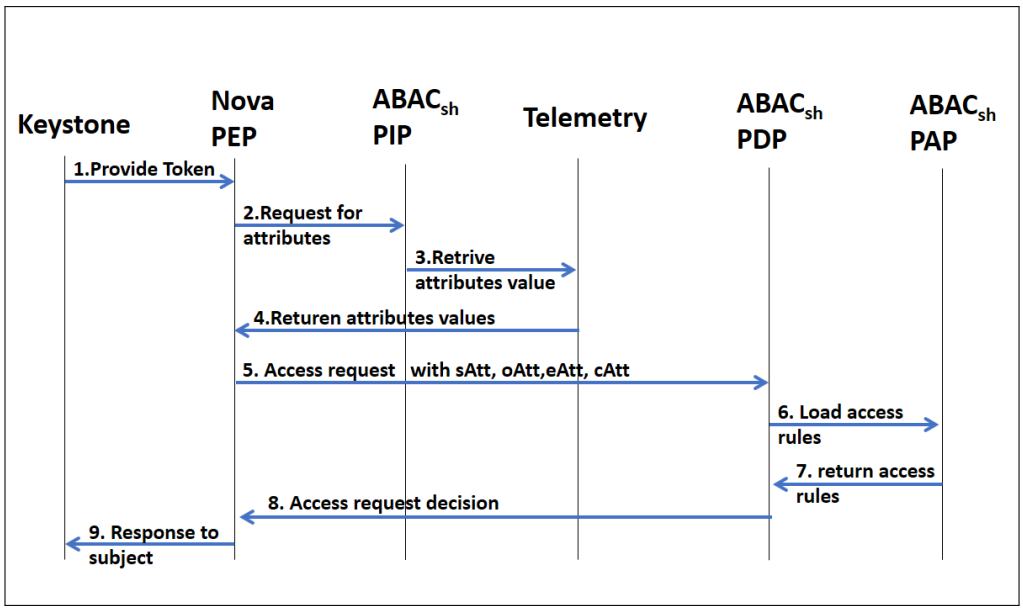

Figure 7.6: proposed $ABAC_{sh}$ for Openstack nova

The proposed $ABAC_{sh}$ enforcement architecture employs the Telemetry service of OpenStack. The telemetry service in Openstack provides the facility to sense the IaaS cloud for environment attributes and the context attributes. The Telemetry service facilitates polling information from the computing service since the proposed access control agent $ABAC_{sh}$ will use the collected information for the attributes assignment process. As an example, the nova service access control process will be used to illustrate $ABAC_{sh}$ extension. Section 7.2.5 introduces the default data flow of nova service access control while Figure 7.6 illustrates nova service access control with $ABAC_{sh}$ extension. The proposed $ABAC_{sh}$ enforcement architecture focuses on three configuration points: PIP (Policy Information Point), PAP (Policy Admission Point) and PDP(Policy Decision Point).

$ABAC_{sh}$ enforcement architecture is divided into three components as follows:

1. `ABAC`$_{sh}$`_PIP`: this is used to collect attributes information from the access control entities, the environment, and the system context. PIP can be achieved in Openstack through configuring the Telemetry component. The Telemetry service is designed to support a billing system by gathering the required information. Therefore, its structure will be beneficial in providing PIP with required attribute information. Telemetry consists of five building blocks: Compute Agent, Central Agent, Collector, Data Store and API Server in order to perform five essential functions [167]. Figure 7.7 summarises the Telemeter process to collect data for further analysis. Telemeter can be configured to collect the attributes and save them in JSON file as this file format is used to store policy rules in OpenStack

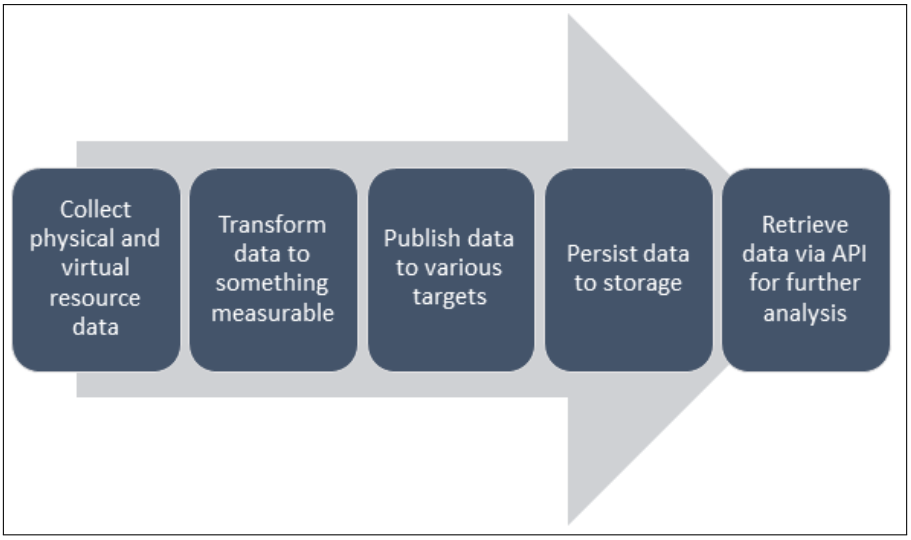

Figure 7.7: Telemeter process

2. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PAP}$: The knowledge database for ABAC_{sh} model consists of access rules from `SoD_rules` and `Policy_rules`. The access rules are created by the system administrator. Those rules will be stored in JSON file format to facilitate its implementation in OpenStack.

3. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$: this is the logical component which reasons about access control in ABACsh. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$ will get an access-request sentence from $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PEP}$ that consist of the attributes information with the access request. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$ will load the access rules from $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PAP}$. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$ accomplishes logical reasoning through forward-chaining algorithm. The result of the logical reasoning indicates if the access is permitted or denied.

7.3.2 Prototype Implementation

The first stage of ABAC_{sh} deployment in Openstack is to be implemented on nova component. $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$ part will be implemented as a prototype.

- **Scope and Assumption**.
  IaaS access control tenant scope can be a single tenant [108], multi-tenant [150, 40, 176] and collaborating parties a cross-clouds [127, 178].

  The implementation scope of access control in this thesis is within single tenant whereas its hypothesis is applicable to multi-tenant and cross-clouds as the big concept behind ABAC_{sh} is user-id free and attributes-based. The proposed ABAC_{sh} is not replacing OpenStack RBAC in this stage. Instead, it allows fine-grained access control and opens prospective avenues to replace RBAC in the near feature.

- **OpenStack Testbed**.

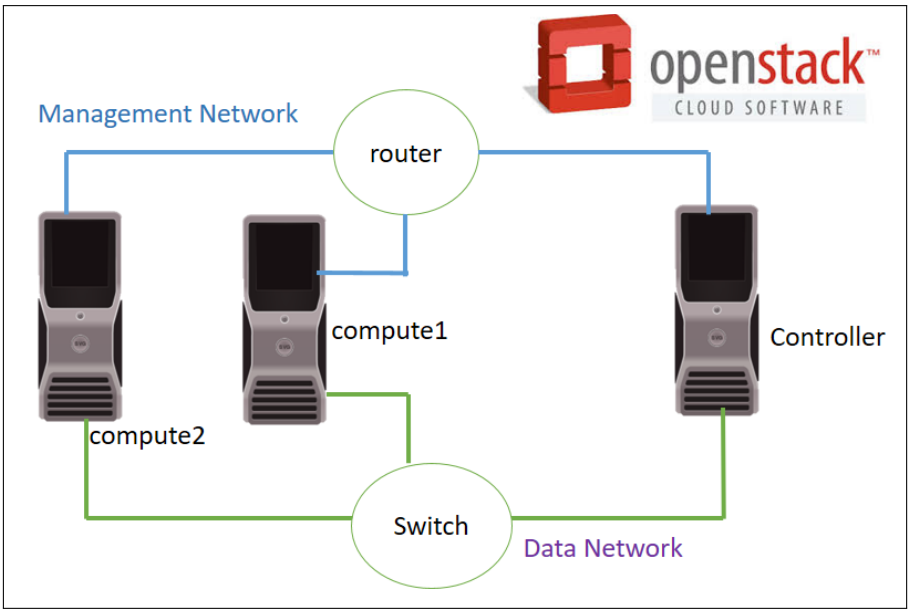

Figure 7.8: OpenStack testbed in physical machines

OpenStack aids in deploying IaaS cloud. Figure 7.8 shows the deployed testbed in this thesis. It is installed in three machines using Ubuntu 16.04 LTS as an operating system and OpenStack-Ocata the latest release (Feb 2017). One machine is configured as a controller which provides OpenStack main server in addition to networking services (neutron), keystone, nova and glance. The Two other machines are configured as compute nodes where virtual machines are hosted. The machines specification is Intel Core i5-4460 CPU Processor 3.20GHz × 4, 15.5 GiB memory, 235 GB Disk and 2 NICs cards. The testbed networking consists of two LANs: management network and data network. The management network traffics the Openstack service communication where data network connects the communication of the virtual machine. This IaaS is a private cloud where OpenStack services and the VMs are accessed by the LAN users.

- **Data flow**.

  Nova policy engine is embedded within its configuration files, therefore it is considered as one of OpenStack's limitations. However, the default policies can be overwritten if policy.json is enabled. As explained in Section 7.2.4, policy.json can be configured to call an external policy engine through URL. The token which was introduced in Section 7.2.4 hold information that can be passed from OpenStack keystone to $ABAC_{sh}$ policy engine via RESET GET-call. Nova PEP receives an access decision from $ABAC_{sh}$ policy engine via RESET POST-call. $ABAC_{sh}$ policy engine use a forward-chaining

algorithm to produce an access control decision. The access control reasoning takes facts which are subject and object attributes, in addition to the system and context attributes. Based on access rules defined by the administrator, the access request will be allowed or denied. Figure 7.9 shows the $ABAC_{sh}$_PDP extension to nova authorization. A policy engine is designed and implemented to add an access control enforcement based on attributes.

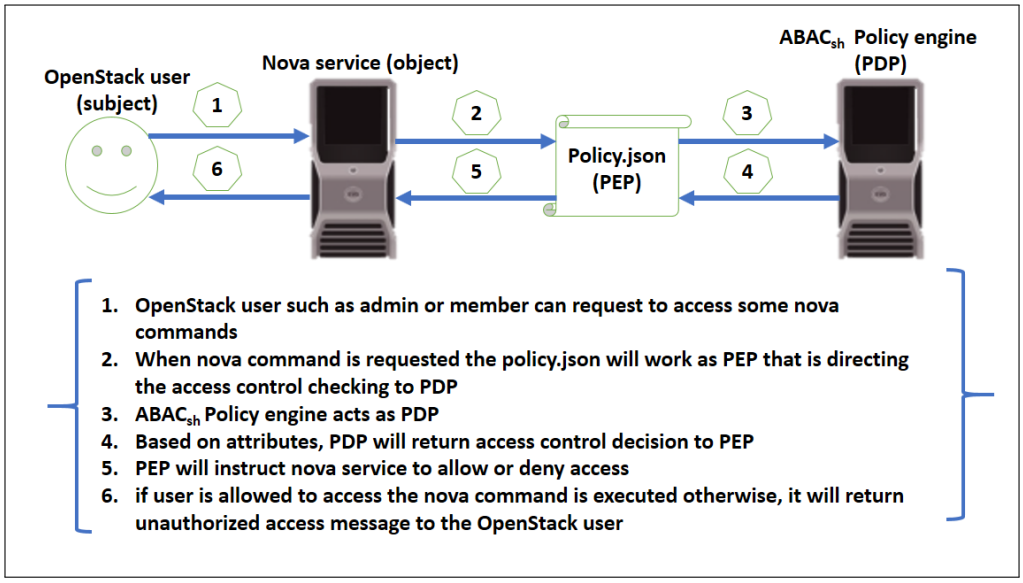

Figure 7.9: The prototype Implementation Data flow

In access control terminology, the Openstack users are the subject, the nova resources are the objects, the policy.json is PEP and ABACsh policy engine is the PDP. The attributes are extracted from the following channels

- The subject attributes can be extracted from keystone token where the available information is `user_name`, `user_id`, `user_password`, `role_id`, `role_name`. The Token information can be retrieved from ' ' content-type:application/json' ' through curl command.
- The extracted nova metrics from the OpenStack system via a command openstack quota show are considered as the object attributes. The attributes information is stored in JSON file format
- The nova environment attributes are extracted via the OpenStack command openstack hypervisor stats show. The attributes information is stored in JSON file format
- The context attributes are not implemented in this prototype but it is visible to be included via Telemetry OpenStack service

$ABAC_{sh}$ policy engine server is implemented using several programming technologies. The web server is developed using Python programming language with web.py since OpenStack services is using python. RESETful API utilities are used to allow the communication between $ABAC_{sh}$ and OpenStack APIs. The forward-reasoning function is programmed using java since this programming language can be smoothly integrated into web programming. To allow the technical interaction between python and java, jpype is used [143, 156]. Data is stored in JSON file formats such as policy data and attributes data.

7.4 Performance Evaluation

Table 7.5: Performance evaluation metrics

Performance metrics element	Description	The applicable Access Control component
response time	The time required to process access request should meet the organization requirement	PEP, PDP, PIP, PRP
Policy repository and retrieval	The repositories form used to store and retrieve the access control policy should balance the cost of hardware and software with the required efficiency by the organization	PAP, PIP, PRP
Policy distribution	If there exist a mechanism that can be used for access control policy distribution	PAP, PIP, PRP
Integrated with authentication function	if the subject and object can be associated with some identifications through an authentication function.	PIP
Key: Policy Decision Point (PDP), Policy Administration Point (PAP), Policy Enforcement Point (PEP), Policy Information Point (PIP), Policy Retrieval Point (PRP)		

The aim of this performance evaluation is to detect if $ABAC_{sh}$ deployment in Openstack introduces any significant overhead. The efficiency of deploying an access control model depends on several factors. The quality of service (QoS) measures can be calculated by performance properties and computation complexity [231, 97]. The computation complexity for $ABAC_{sh}$ has been discussed in Section 3.4. In this section, the performance metrics are evaluated.

The performance metrics consist of four elements: response time, policy repository and retrieval, policy distribution, and Integrated with authentication function [97]. Table 7.5 explains theses performance metrics elements and on which access control components they can be applied. Since the implemented prototype

is the $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$, the followed experiments will measure response time. With regard to policy repository and retrieval, the implemented $ABAC_{sh}$ use JSON file to store access control policy which does not add any extra hardware or software cost to OpenStack IaaS cloud as this form of policy storage is used by OpenStack. The remaining two performance metrics elements are not calculated in this stage as PIP is not implemented.

7.4.1 Experiment Content

In this section, the performance evaluation of the implemented $ABAC_{sh}$ prototype in OpenStack is presented. Specifically, $ABAC_{sh}$ policy engine which represents PDP of access control model is discussed. The experiments fall into three parts where the response time is calculated. Response time indicates the time consumed by the system in order to process the access request decision call. The response time has been used to measure the performance in several OpenStack implementations such as in [50, 207]. In these experiments, OpenStack cloud was installed in physical servers running Ubuntu 16.04 LTS release.

Three types of execution time can be measured [136, 17]. The first one is real-time that reflects the wall clock where the time is calculated from the start till the end of the call including the waiting time and time used by other processes. The second one is user-time that reflects the actual CPU-time spent outside the kernel during the process call in user-mode without considering other processes. The third one is sys-time that reflects the actual CPU-time spent within the kernel during the process execution. Three experimental settings have been implemented as explained below.

- **Experiment 1 (Exp1):** The response time for the default access control model to process access request to nova resources. The default use RBAC and Oslo policy engine.
- **Experiment 2 (Exp2):** The response time for extending the default nova policy engine with $ABAC_{sh}$ that utilizes 24 attributes in access control processing
- **Experiment 3 (Exp3):** The response time for extending the default nova policy engine with $ABAC_{sh}$ that use forward-chaining for access control reasoning.

7.4.2 Experimental Results and Discussion

Each experiment was run five times, and then the average value was recorded. Five scenarios were observed by increasing the number of requests from five to

Table 7.6: Experimental Results

no. of Requests	Response time	Exp1	Exp2	Exp3
5	real	8.67	8.96	8.77
	user	5.56	5.52	5.55
	sys	0.42	0.44	0.40
10	real	17.24	17.34	17.24
	user	10.99	11.05	11.11
	sys	0.88	0.86	0.83
15	real	25.52	26.08	25.87
	user	16.46	16.61	16.64
	sys	1.25	1.31	1.22
20	real	34.40	34.56	34.45
	user	22.07	22.17	22.23
	sys	1.68	1.66	1.56
25	real	43.02	43.05	43.07
	user	27.64	27.63	27.82
	sys	2.09	2.10	1.96

twenty-five as illustrated in Table 7.6. The request indicates an access control request from a user(subject) to access nova-resources(object).

Based on usability engineering [154, 209] The response time value can be within three categories: over 0.1 seconds will give the user a feeling that the system is reacting instantaneously, over 1.0 seconds will give the user a feeling of a delay but will stay uninterrupted, over 10 seconds the user will lose his/her attention and will search for something to work on till the computer responds.

Three time values has been recorded as illustrated in Table 7.6: real-time, user-time and sys-time. In this study, real-time and sys-time have a direct effect on the performance analysis whereas user-time is reflecting the processing outside the kernel. The real-time shows the access control execution time in additions to the other OpenStack cloud processes that introduce some delay by blocking the process or introducing a waiting time. Therefore, this measurement will indicate the effect of our extended $ABAC_{sh}$ nova to the overall OpenStack system.

The graph in Figure 7.10 compares the real-time for the three set of experiments. The increase is 0.05 seconds when the extended $ABAC_{sh}$ nova employ forward reasoning in access decision processing as shown in Table 7.7. While the increase is 0.145 seconds when $ABAC_{sh}$ uses twenty-four attributes in access decision processing. Therefore, there is an increase of 0.56% when attributes are added to the policy engine and 0.19% when the forward-chaining algorithm is added. Consequently, the increase in response time is negligible in reference to the usability engineering when the nova default access control is extended with part of the proposed ABAC enhancement.

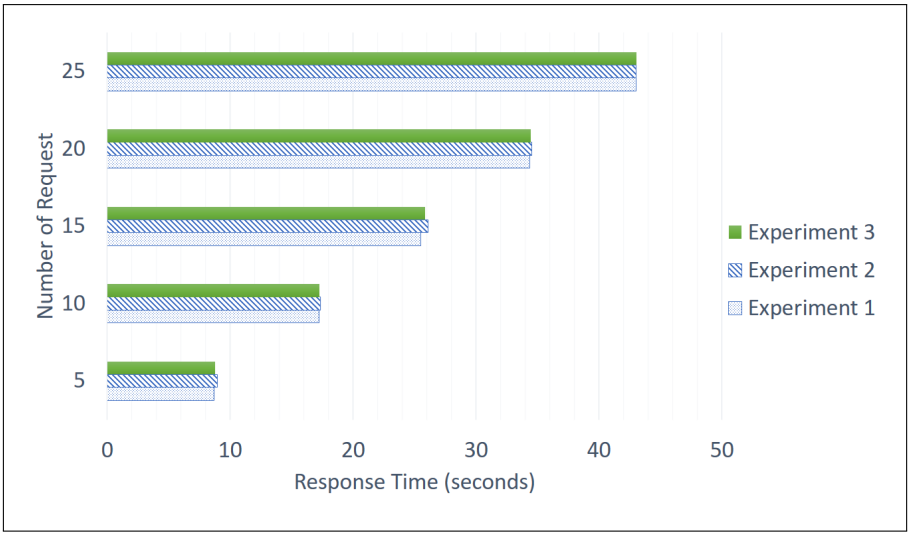

Figure 7.10: real-time for access control processing in nova

Table 7.7: Comparing real-time values

Experiment 2 - Experiment 1	Experiment 3 - Experiment 1
0.29 0.1 0.56 0.16 0.03	0.1 0 0.35 0.05 0.05
average 0.145	average 0.05
percentage 0.56%	percentage 0.19%

On the other hand, sys-time gives the process execution only within the kernel regardless of the other tasks. Therefore, the time for the 25 requests dropped from 43.02 seconds within real-time to 2.09 seconds within sys-time during Exp1 which involve default nova access control. The sys-time comparison for the three experiments is illustrated in Figure 7.11. The results show a slightly better performance of 5.5% for extending the default nova access control when forward-reasoning has been utilized whereas an increase of 4% over the default nova when 24-attributes are used in the policy-engine as illustrated in Table 7.8.

From these results, the $ABAC_{sh}$ shows an acceptable performance compared to the default OpenStack access control within nova service.This section demonstrates the enhanced attribute-based access control $ABAC_{sh}$ performance improvement when attributes and forward reasoning algorithm are employed. It has been noticed that the performance improvement is liner in Figure 7.10 when only attributes are involved in access decision. Whereas in Figure 7.11 when forward reasoning is involved, an improvement in performance has been noticed. This in-

dicates an opportunity of enhancing the IaaS-cloud security when logical reasoning and AI mechanism are involved in access control system.

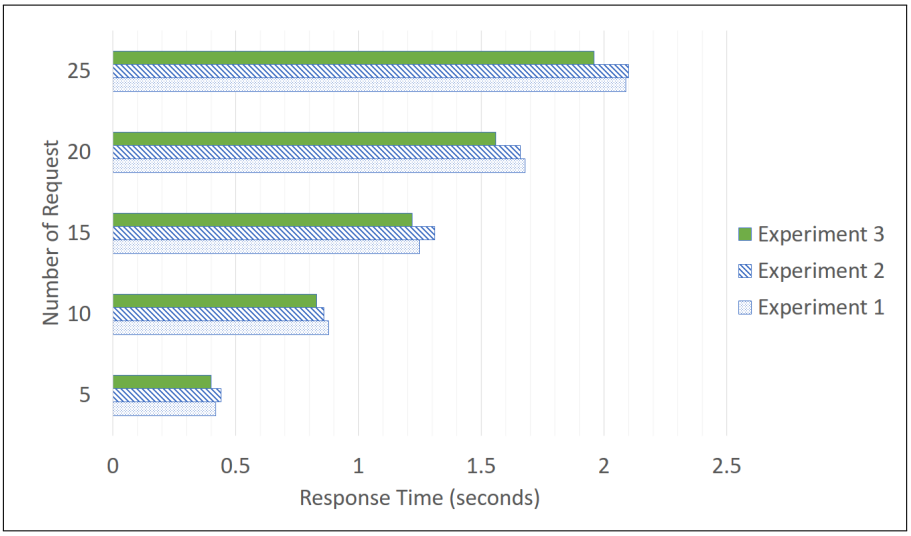

Figure 7.11: sys-time for access control processing in nova

Table 7.8: Comparing sys-time values

Experiment 2 - Experiment 1	Experiment 3 - Experiment 1
0.02 -0.02 0.06 -0.02 0.01	-0.02 -0.05 -0.03 -0.12 -0.13
average 0.05	average -0.07
percentage 4%	percentage -5.5%

7.4.3 Limitations of Experiment

The main aim of the experiments in this chapter is to study the performance improvement when attribute based access control model is introduced into IaaS cloud. The experiment scale is limited to a private cloud in a LAN set-up. Therefore, the network performance metrics has not been studied such as the latency and throughput.

The implementation in this thesis does not involve the PIP component of the access control, therefore only a simple forward reasoning algorithm has been deployed without knowledge update component. The used database for knowledge is written manually whereas the system should use an automated information collection method if PIP is implemented. One subject is involved in each experiment,

therefore multi-access has not been investigated in this thesis. Multi-tenant study is a critical future work.

7.5 Comparative Study of ABAC Models Implementation in OpenStack

OpenStack IaaS-cloud is open-source and several recognized bodies are contributing to its development such as NASA, Rackspace, RedHat, CISCO and many others listed in OpenStack contributors [165].

Table 7.9: Compare the $ABAC_{sh}$ implementation in OpenStack with related work

	[152]	[25]	[109]	Proposed $ABAC_{sh}$
PEP	OpenStack services (Nova, Glance ,,,)			
architecture	client-server based (RESTful API)			
PDP	Based on Prevention-Based Access Control (PBAC)	Based on Policy Machine (PM) mechanism within OpenStack LAN	Based on external Policy engine service outside OpenStack LAN	Based on external Policy engine service within OpenStack LAN
ABAC model	None	Role-centric based model with user attributes	ABACα	Proposed $ABAC_{sh}$
Evaluation metrics	Execution time in sec,Exp1: without PBAC,Exp2: 20 Edges,Exp3: 1000 Edges	Performance: Time in sec/no. request, `OS_RBAC`, `OS_PM_RBAC`, `OS_PM_ABAC`	Performance: Tim in sec/no. user and no. request attributes values, 0=RBAC, 5,10,15,20 attributes	Performance: Time in sec/no. request, 0 attributes and default OSAC, 24 attributes with the ABAC extension,24 attributes with forward-chaining algorithm

OpenStack platform is flexible and allows the developers to modify the source code. Riad in [179] focused on studying the type and the number of attributes required in ABAC. He implemented a prototype in OpenStack and discusses the average time that is taken by keystone to generate a token when attributes are included and when it is excluded. Ibrahim in [159] discusses an XACML extension for Openstack but without practical evaluation. The most related work to this

research is summarized in Table 7.9 with a comparison to the proposed $ABAC_{sh}$ extension. In contrast to [152], the proposed work is a type of ABAC model where [152] is based on Prevention data concept. The work done by [25] concentrated on user attributes whereas the proposed $ABAC_{sh}$ involve four types of attributes since ABAC model is meant to involve attributes of the subject, the object, the environment and the context based on the literature review. Moreover, in [25], the process of assigning values to user attributes are not explained as it is out of their research scope. Therefore, this thesis contributes in giving details of attributes value assignment in $\texttt{ABAC}_{\texttt{sh}}\texttt{\_PIP}$ configuration point via Telemetry service. The accomplished evaluation by [109] focuses on the performance when attributes are added to the token within keystone, while in $ABAC_{sh}$ the focus was on the performance of the complete access control process from getting the access request till returning the access decision.

7.6 Summary and Discussion

In this chapter, a demonstration of the enhanced attribute-based access control $ABAC_{sh}$ implementation is presented. Openstack platform is used to deploy IaaS cloud since it is open-source and supports agent architecture. The OpenStack platform has been extended to implement $ABAC_{sh}$ by adding an external policy engine. An enforcement architecture is designed, and a prototype is implemented. The enforcement architecture use Telemeter OpenStack component to implement agent-structure to collect the required information for access control process.

The implemented prototype includes and extend policy engine that utilises forward-chaining algorithm to reason about access decision and adds attribute values for the object. The performance evaluation consist of three experiments. The Experiment1 involves the default OpenStack access control. The Experiment2 involves $ABAC_{sh}$ with 24-attributes value for object. The Experiment3 involves $ABAC_{sh}$ with policy-engine that is deploying forward-chaining. The experimental results shows a negligible increase in response time for real-time and system-time as illustrated in Table 7.7 and Table 7.8. Based on usability engineering, if the response time value reach 0.1 seconds, then the user will feel that the system is reacting instantaneously. The average increase in response time does not exceed 0.15 for all cases which indicates that the increase in response time is negligible.

This implementation can be considered as an initial step in deploying ABAC model in IaaS cloud access control. There is a wide space to extend and improve this implementation such as:

- Deploying PIP via OpenStack telemetry component

- Study multi-tenancy scenarios by involving multi-subject.
- Scenarios to study the forward reasoning algorithm within the best and the worst case.

Chapter 8

Conclusions and Future Work

This thesis investigates the security challenges facing IaaS cloud and focuses on the access control model. The IaaS access control requires dynamic and fine-grained features. Therefore, this thesis proposes an intelligent attribute based access control model and discusses its implementation. Section 8.1 summarises the thesis contributions, and Section 8.2 describes potential future work.

8.1 Conclusion

IaaS cloud provides dynamic, elastics, multi-tenant and easy to configure computing infrastructure. It is constructed based on several technologies such as virtualisation and grid computing. Besides the benefits gained from IaaS, there are several challenges such as the security aspects. This thesis focusses on enhancing IaaS cloud security from access control perspective. Due to the IaaS dynamic environment characteristics and the flexibility for the users to join and leave the system, there is a need for an access control model that is able to cope with this environment. The literature relating to IaaS cloud access control reveal an approach to introduce attribute-based access control model into cloud computing. Attribute-based access control (ABAC) faces two main challenges to be deployed in IaaS cloud. The first one is related to formal ABAC specification and reasoning. The second one is related to the specification of its conceptual characteristics.

This thesis contributes to the IaaS cloud security enhancement by introducing an attribute-based access control model that meets IaaS cloud characteristics. Furthermore, an artificial intelligence framework is designed and implemented in OpenStack testbed. The enhanced attribute-based access control (ABACsh) is designed to be context-aware and supports SoD, as discussed in Chapter 3. To the best knowledge of the author, there is no existing formal logic that is able to describe, verify and reason about ABAC models; this thesis introduces an Ac-

cess Control Logic (ACL) combined with Deontic Logic (DL) that can satisfy this requirement, as demonstrated in Chapter 4. The proposed ACL-DL is used to formally specify ABACsh and logical infer its access decision, as explained in Chapter 5. The formal specification of an access control is a critical necessity for facilitation of the implementation stage. An intelligent framework is proposed to support ABACsh dynamic feature as introduced in Chapter 6. An implementation of a prototype of this framework is illustrated in Chapter7 with performance analysis. The contributions of this thesis are summarised in the table 8.1

The understanding of IaaS access-control requirements is essential to study the improvement opportunities in cloud security. The access control in IaaS is still an evolving area as the current implemented model is RBAC whereas ABAC is recommended model by several researchers. Moreover, applications that involve artificial intelligent and formal logic is a critical research that allows a significant improvement in security systems. The modal logic field can contribute in reasoning about knowledge which supports the development of expert systems. The link between these areas can open critical research directions.

8.2 Future Work

This research can open further studies and applications. ABACsh can be improved by adding support for access control concepts such as delegation and object-oriented concept such as hierarchy. Take into consideration that ABAC models should be ID-free and based on attributes to maintain its flexibility and dynamic features. Further study for ACL-DL can be conducted using a theorem prover, such as NuSmv, to implement an automated reasoning tool. The proposed ABACsh enforcement architecture can be deployed via Telemeter component in OpenStack, where a Heat component can be used in the implementation phase to construct an intelligent ABACsh that can be employed in a real IaaS cloud operating environment.

Table 8.1: Summary of the Thesis Contributions

Chapter no.	Main contribution	Details	objective
2	Analytical study to identify access control challenges and requirements in IaaS cloud	•Study the security challenges related to access control in IaaS. •Identify IaaS access control requirements. •Identify the approaches that can contribute to enhancing IaaS access control systems.	Obj.1, Obj.2
3	An enhanced attribute based access control $ABAC_{sh}$	•Context-aware. •Support SoD	Obj.3
4	A formal logic that is able to specify $ABAC_{sh}$, known as Access Control Logic Deontic Logic (ACL-DL)	•Formal logic language. •Kripke structure. •ACL-DL Formal verification	Obj.4
5	A formal specification and logical reasoning for $ABAC_{sh}$ using ACL-DL	•Formal description of the $ABAC_{sh}$ states, relations, and functions. •Logical reasoning about $ABAC_{sh}$. •Explained through a case study	Obj.4
6	An intelligent framework for $ABAC_{sh}$	•A framework that is based on agent and logic. •Utilise knowledge-agent implemented through rule-based expert system mechanisms.	Obj.5
7	$ABAC_{sh}$ implementation in IaaS cloud testbed (OpenStack)	•show the visibility of the proposed intelligent framework for $ABAC_{sh}$. •An enforcement architecture for the $ABAC_{sh}$ agent is designed using OpenStack components. •$\texttt{ABAC}_{\texttt{sh}}\texttt{\_PDP}$ is deployed in OpenStack	Obj.6

References

[1] Younis A. Younis, Kashif Kifayat, and Madjid Merabti. An access control model for cloud computing. *Journal of Information Security and Applications*, 19(1):45–60, feb 2014.

[2] Martín Abadi. Variations in Access Control Logic. pages 96–109. Springer, Berlin, Heidelberg, 2008.

[3] Martín Abadi. Logic in Access Control (Tutorial Notes). pages 145–165. Springer Berlin Heidelberg, 2009.

[4] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, sep 1993.

[5] Anas Abou El Kalam. A research challenge in modeling access control policies: Modeling recommendations. In *2008 Second International Conference on Research Challenges in Information Science*, pages 263–270. IEEE, jun 2008.

[6] Thomas. Agotnes, Jan. Broersen, and Dag. Elgesem. *Deontic logic in computer science : 11th International Conference, DEON 2012, Bergen, Norway, July 16-18, 2012. Proceedings*. Springer, 2012.

[7] Dana Al Kukhun. *Steps towards adaptive situation and context-aware access: a contribution to the extension of access control mechanisms within pervasive information systems*. Phd, Research Institute in Computer Science of Toulouse, 2012.

[8] Shadha ALAmri and Lin Guan. Exploring the Firewall Security Consistency in Cloud Computing during Live Migration. In *7th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, volume 06-08-July, pages 1–6, New York, New York, USA, 2016. ACM digital library.

[9] Shadha ALAmri and Lin Guan. Infrastructure as a service: Exploring network access control challenges. In *2016 SAI Computing Conference (SAI)*, pages 596–603. IEEE, jul 2016.

[10] Suhair Alshehri. *Toward Effective Access Control Using Attributes and Pseudoroles*. PhD thesis, 2014.

[11] Hasan Amjad. Combining model checking and theorem proving. Technical report, University of Cambridge, 2004.

[12] Chaker El Amrani, Kaoutar Bahri Filali, Kaoutar Ben Ahmed, Amadou Tidiane Diallo, Stephano Telolahy, and Tarek El-Ghazawi. A Compartive Study of Cloud Computing Middleware. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 690–693, Washington, 2012. IEEE Computer Society.

[13] M. R. Anala, Jyoti Shetty, and G. Shobha. A framework for secure live migration of virtual machines. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 243–248. IEEE, aug 2013.

[14] Bayu Anggorojati, Neeli Rashmi Prasad, and Ramjee Prasad. Secure capability-based access control in the M2M local cloud platform. In *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE)*, pages 1–5. IEEE, may 2014.

[15] G.Aldo Antonelli. A directly cautious theory of defeasible consequence for default logic via the notion of general extension. *Artificial Intelligence*, 109(1-2):71–109, jun 1999.

[16] Grigoris Antoniou, Matteo Baldoni, Piero A. Bonatti, Wolfgang Nejdl, and Daniel Olmedilla. Rule-based Policy Specification. In *Secure Data Management in Decentralized Systems*, pages 169–216. Springer US, Boston, MA, 2007.

[17] Askubuntu.com. command line - How can I measure the execution time of a terminal process? - Ask Ubuntu, 2011.

[18] Guillaume Aucher, Guido Boella, and Leendert van der Torre. A dynamic logic for privacy compliance. *Artificial Intelligence and Law*, 19(2-3):187–231, aug 2011.

[19] R. K. Banyal, V. K. Jain, and Pragya Jain. Dynamic Trust Based Access Control Framework for Securing Multi-Cloud Environment. In *Proceedings of the 2014 International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '14*, pages 1–8, New York, New York, USA, nov 2014. ACM Press.

[20] Lujo Bauer and Florian Kerschbaum. What are the most important challenges for access control in new computing domains, such as mobile, cloud and cyber-physical systems? In *Proceedings of the 19th ACM symposium on Access control models and technologies - SACMAT '14*, pages 127–128, New York, New York, USA, jun 2014. ACM Press.

[21] Matthias Beckerle and Leonardo A. Martucci. Formal definitions for usable access control rule sets from goals to metrics. In *Proceedings of the Ninth Symposium on Usable Privacy and Security - SOUPS '13*, page 1, New York, New York, USA, 2013. ACM Press.

[22] Pratap Kumar Behera and Pabitra Mohan Khilar. A Novel Trust Based Access Control Model for Cloud Environment. In *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*, pages 285–295. Springer, New Delhi, 2017.

[23] M. Beirlaen and C. Strasser. Non-monotonic reasoning with normative conflicts in multi-agent deontic logic. *Journal of Logic and Computation*, 24(6):1179–1207, dec 2014.

[24] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, feb 2003.

[25] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. An Attribute-Based Access Control Extension for OpenStack and Its Enforcement Utilizing the Policy Machine. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 37–45. IEEE, nov 2016.

[26] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Mitigating Multi-Tenancy Risks in IaaS Cloud Through Constraints-Driven Virtual Resource Scheduling. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies - SACMAT '15*, pages 63–74, New York, New York, USA, jun 2015. ACM Press.

[27] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Virtual Resource Orchestration Constraints in Cloud Infrastructure as a Service. In *Proceedings of*

the 5th ACM Conference on Data and Application Security and Privacy - CODASPY '15, pages 183–194, New York, New York, USA, mar 2015. ACM Press.

[28] Khalid Zaman Bijon, Ram Krishnan, and Ravi Sandhu. Towards an Attribute Based Constraints Specification Language. In *2013 International Conference on Social Computing*, pages 108–113. IEEE, sep 2013.

[29] Khalid Zamon Bijon. *Constraints for attribute based access control with application in cloud IaaS*. PhD thesis, THE UNIVERSITY OF TEXAS AT SAN ANTONIO, 2015.

[30] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. Label-Based Access Control. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16*, pages 1–12, New York, New York, USA, mar 2016. ACM Press.

[31] Patrick Blackburn, Maarten de. Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, 2001.

[32] Patrick Blackburn and Johan van Benthem. Modal logic: a semantic perspective. In *Foundations of Software Science and Computational Structures*, pages 1–84. 2007.

[33] Philogene A. Boampong and Luay A. Wahsheh. Different facets of security in the cloud. page 5, mar 2012.

[34] Guido Boella, Dov M. Gabbay, Valerio Genovese, and Leendert van der Torre. Fibred Security Language. *Studia Logica*, 92(3):395–436, aug 2009.

[35] R. A. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[36] Pascal Bouvry and El-Ghazali Talbi. Computational Intelligence for Cloud Computing [Guest Editorial]. *IEEE Computational Intelligence Magazine*, 10(1):18–51, feb 2015.

[37] Julien Bringer, Beatriz Gallego, Ghassan Karame, Mathias Kohler, Panos Louridas, Melek Önen, Hubert Ritzdorf, Alessandro Sorniotti, and David Vallejo. TREDISEC: Trust-Aware REliable and Distributed Information SEcurity in the Cloud. In *E-Democracy Citizen Rights in the World of the New Computing Paradigms*, pages 193–197. Springer International Publishing, 2015.

[38] Jan Broersen, Stephen Cranefield, Yehia Elrakaiby, Dov Gabbay, Davide Grossi, Emiliano Lorini, Xavier Parent, Leendert W. N. van der Torre, Luca Tummolini, Paolo Turrini, and François Schwarzentruber. Normative Reasoning and Consequence. *DROPS-IDN/3999*, 4, 2013.

[39] Julien Brunel, Jean-Paul Bodeveix, and Mamoun Filali. A State/Event Temporal Deontic Logic. In *Deontic Logic and Artificial Normative Systems*, pages 85–100. Springer, Berlin, Heidelberg, 2006.

[40] Daniel J. Buehrer and Chun-Yao Wang. CA-ABAC: Class Algebra Attribute-Based Access Control. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 220–225. IEEE, dec 2012.

[41] Rajkumar Buyya, James Broberg, Andrzej M Goscinski, Benny Rochwerger, Constantino Vázquez, David Breitgand, David Hadas, Massimo Villari, Philippe Massonet, Eliezer Levy, Alex Galis, Ignacio M. Llorente, Rubén S. Montero, Yaron Wolfsthal, Kenneth Nagin, Lars Larsson, and Fermín Galán. *Cloud computing: Principles and paradigms*, volume 87. 2010.

[42] Laurent Catach. TABLEAUX: A general theorem prover for modal logics. *Journal of Automated Reasoning*, 7(4):489–510, dec 1991.

[43] Sol Cates. The evolution of security intelligence. *Network Security*, 2015(3):8–10, mar 2015.

[44] A Cavoukian, M Chibba, G Williamson, and A Ferguson. The Importance of ABAC: Attribute-Based Access Control to Big Data: Privacy and Context. *Privacy and Big Data Institute, Ryerson University, Toronto, Canada*, 2015.

[45] J. Cheng and J. Miura. Deontic relevant logic as the logical basis for specifying, verifying, and reasoning about information security and information assurance. In *First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8 pp.–608. IEEE, 2006.

[46] Jingde Cheng, Shinsuke Nara, and Yuichi Goto. FreeEnCal: A Forward Reasoning Engine with General-Purpose. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 444–452. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[47] SK Chin and SB Older. *Access Control, Security, and Trust: A Logical Approach*. CRC press, 2010.

[48] Chang Choi, Junho Choi, and Pankoo Kim. Ontology-based access control model for security policy reasoning in cloud computing. *The Journal of Supercomputing*, 67(3):711–722, 2014.

[49] L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, pages 103–112. IEEE Comput. Soc. Press, 1997.

[50] Antonio Corradi, Mario Fanelli, and Luca Foschini. VM consolidation: A real case based on OpenStack Cloud. *Future Generation Computer Systems*, 32:118–127, mar 2014.

[51] Agostino Cortesi, Gilberto File, and Bernhard Steffen. *Static analysis : 6th international symposium, SAS'99, Venice, Italy, September 22-24, 1999 : proceedings*. Springer, 1999.

[52] Ed Coyne and Timothy R. Weil. ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. *IT Professional*, 15(3):14–16, may 2013.

[53] Jason Crampton and Conrad Williams. On Completeness in Languages for Attribute-Based Access Control. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies - SACMAT '16*, pages 149–160, New York, New York, USA, 2016. ACM Press.

[54] Jordan Cropper, Johanna Ullrich, Peter Fruhwirt, and Edgar Weippl. The Role and Security of Firewalls in IaaS Cloud Computing. In *2015 10th International Conference on Availability, Reliability and Security*, pages 70–79. IEEE, aug 2015.

[55] Nora Cuppens-Boulahia and Frédéric Cuppens. Specifying Intrusion Detection and Reaction Policies: An Application of Deontic Logic. In *Deontic Logic in Computer Science*, pages 65–80. Springer, Berlin, Heidelberg, 2008.

[56] Carlos Viegas Damásio, Anastasia Analyti, Grigoris Antoniou, and Gerd Wagner. Supporting Open and Closed World Reasoning on the Web. In *Principles and Practice of Semantic Web Reasoning*, pages 149–163. Springer, Berlin, Heidelberg, 2006.

[57] N Damianou, A Bandara, and M Sloman. A survey of policy specification approaches. *Formal Aspects of Security and Trust, Springer*, 2002.

[58] Boris Danev, Ramya Jayaram Masti, Ghassan O. Karame, and Srdjan Capkun. Enabling secure VM-vTPM migration in private clouds. In *Proceedings*

of the 27th Annual Computer Security Applications Conference on - ACSAC '11, page 187, New York, New York, USA, dec 2011. ACM Press.

[59] Cerri Davide, Corcoglioniti Francesco, Kopecky Jacek, Lafite Michael, Teymourian Kia, and Toro del Valle German. European Commission : CORDIS : Projects & Results Service : Triple space communication. Technical report, The European Commission, 2009.

[60] Wanderson Paim de Jesus, Daniel Alves da Silva, Rafael T. de Sousa Junior, and Francisco Vitor Lopes da Frota. Analysis of SDN Contributions for Cloud Computing Security. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 922–927. IEEE, dec 2014.

[61] Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky. Permission to speak: A logic for access control and conformance. *The Journal of Logic and Algebraic Programming*, 80(1):50–74, 2011.

[62] Frank Doelitzscher. *Security Audit Compliance for Cloud Computing*. Doctorate, 2014.

[63] Frank Doelitzscher, Christoph Reich, Martin Knahl, Alexander Passfall, and Nathan Clarke. An agent based business aware incident detection system for cloud environments. *Journal of Cloud Computing: Advances, Systems and Applications*, 1(1):9, jul 2012.

[64] Qi Duan and Ehab Al-Shaer. Traffic-aware dynamic firewall policy management: techniques and applications. *IEEE Communications Magazine*, 51(7):73–79, jul 2013.

[65] Robert Dukaric and Matjaz B. Juric. Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29(5):1196–1210, jul 2013.

[66] Adrian Duncan, Sadie Creese, Michael Goldsmith, and Jamie S. Quinton. Cloud Computing: Insider Attacks on Virtual Machines during Migration. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 493–500. IEEE, jul 2013.

[67] Thomas Erl, Richardo Ricardo Richardo Puttini, and Zaigham Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, may 2013.

[68] David Evans and David M. Eyers. Deontic logic for modelling data flow and use compliance. In *Proceedings of the 6th international workshop on*

Middleware for pervasive and ad-hoc computing - MPAC '08, pages 19–24, New York, New York, USA, 2008. ACM Press.

[69] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning about knowledge*. MIT Press, 2003.

[70] Arjumand Fatima, Yumna Ghazi, Muhammad Awais Shibli, and Abdul Ghafoor Abassi. Towards Attribute-Centric Access Control: an ABAC versus RBAC argument. *Security and Communication Networks*, 2016.

[71] Diogo A. B. Fernandes, Liliana F. B. Soares, João V. Gomes, Mário M. Freire, and Pedro R. M. Inácio. Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170, sep 2014.

[72] Diogo AB Fernandes, M'io M Freire, Paulo AP Fazendeiro, and Pedro RM In'cio. Applications of artificial immune systems to computer security: A survey. *Journal of Information Security and Applications*, 35:138–159, aug 2017.

[73] David Ferraiolo. Towards an ABAC Family of Models. *National Institute of Standards and Technology*, 2013.

[74] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The Policy Machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

[75] David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn, and Vincent Hu. Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control - ABAC '16*, pages 13–24, New York, New York, USA, 2016. ACM Press.

[76] David F. DF Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, aug 2001.

[77] Tom Fifield, Diane Fleming, Anne Gentle, Lorin Hochstein, Jonathan Proulx, Everett Toews, and Joe Topjian. OpenStack Docs: OpenStack Documentation Contributor Guide, 2017.

[78] John Franco. KRIPKE MODELS.

[79] Fakher Ben Ftima, Kamel Karoui, and Henda Ben Ghzela. A secure mobile agents approach for anomalies detection on firewalls. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services - iiWAS '08*, page 689, New York, New York, USA, nov 2008. ACM Press.

[80] DM Gabbay. *Fibring logics*. Clarendon Press, 1998.

[81] Deepak Garg. Principal-centric reasoning in constructive authorization logic. Technical report, DTIC Document, 2009.

[82] Deepak Garg and Martín Abadi. A Modal Deconstruction of Access Control Logics. In *Foundations of Software Science and Computational Structures*, pages 216–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[83] Valerio Genovese and Deepak Garg. New Modalities for Access Control Logics: Permission, Control and Ratification. In *Security and Trust Management*, pages 56–71. Springer, Berlin, Heidelberg, 2012.

[84] Valerio Genovese, Daniele Rispoli, Dov M. Gabbay, and Van Der Torre Leendert. Modal Access Control Logic Axiomatization, Semantics and FOL Theorem Proving. In *Proceedings of the 2010 conference on STAIRS 2010: Proceedings of the Fifth Starting AI Researchers' Symposium*, page 369. IOS Press, 2011.

[85] Janice Glasgow, Glenn Macewen, and Prakash Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, aug 1992.

[86] V.D. Gligor, S.I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*, pages 172–183. IEEE Comput. Soc, 1998.

[87] Fouad Guenane, Hajer Boujezza, Michele Nogueira, and Guy Pujolle. An architecture to manage performance and reliability on hybrid cloud-based firewalling. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–5. IEEE, may 2014.

[88] Rémy Haemmerlé and Rémy. On Combining Backward and Forward Chaining in Constraint Logic Programming. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming - PPDP '14*, pages 213–224, New York, New York, USA, 2014. ACM Press.

[89] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, aug 1976.

[90] Reuven Harrison. Reducing complexity in securing heterogeneous networks. *Network Security*, 2015(10):11–13, oct 2015.

[91] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):5, 2013.

[92] Xiangjian He, Thawatchai Chomsiri, Priyadarsi Nanda, and Zhiyuan Tan. Improving cloud network security using the Tree-Rule firewall. *Future Generation Computer Systems*, 30:116–126, jan 2014.

[93] Timothy L. Hinrichs, William C. Garrison, Adam J. Lee, Skip Saunders, and John C. Mitchell. TBA : A Hybrid of Logic and Extensional Access Control Systems. pages 198–213. Springer Berlin Heidelberg, 2012.

[94] Chung Tong Hu, David F. Ferraiolo, and David R. Kuhn. *Assessment of Access Control Systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

[95] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. Detecting and Resolving Firewall Policy Anomalies. *IEEE Transactions on Dependable and Secure Computing*, 9(3):318–331, may 2012.

[96] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. FLOWGUARD. In *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pages 97–102, New York, New York, USA, aug 2014. ACM Press.

[97] VC Hu and KA Kent. *Guidelines for access control system evaluation metrics*. 2012.

[98] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Technical report, NIST, 2014.

[99] Jingwei Huang, David M. Nicol, Rakesh Bobba, and Jun Ho Huh. A framework integrating attribute-based policies into role-based access control. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies - SACMAT '12*, page 187, New York, New York, USA, 2012. ACM Press.

[100] Wei Huang, Afshar Ganjali, Beom Heyn Kim, Sukwon Oh, and David Lie. The State of Public Infrastructure-as-a-Service Cloud Security. *ACM Computing Surveys*, 47(4):1–31, jun 2015.

[101] G. E. (George Edward) Hughes and M. J. Cresswell. *A new introduction to modal logic*. Routledge, 1996.

[102] Michael Huth and Mark Ryan. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press, 2004.

[103] Iliana Iankoulova and Maya Daneva. Cloud computing security requirements: A systematic review. In *2012 Sixth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–7. IEEE, may 2012.

[104] Stuart Jacobs. *Engineering information security : the application of systems engineering concepts to achieve information assurance*. 2011.

[105] Yosr Jarraya, Arash Eghtesadi, Sahba Sadri, Mourad Debbabi, and Makan Pourzandi. Verification of Firewall Reconfiguration for Virtual Machines Migrations in the Cloud. *Computer Networks*, oct 2015.

[106] S. Jha, M. Tripunitara, and W. Winsborough. Towards Formal Verification of Role-Based Access Control Policies. *IEEE Transactions on Dependable and Secure Computing*, 5(4):242–255, oct 2008.

[107] Sadhana Jha, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya. Enforcing Separation of Duty in Attribute Based Access Control Systems. In *Information Systems Security*, pages 61–78. Springer, Cham, 2015.

[108] Xin Jin. *Attribute-based access control models and implementation in cloud infrastructure as a service*. Phd, THE UNIVERSITY OF TEXAS AT SAN ANTONIO, 2014.

[109] Xin Jin, Ram Krishnan, and Ravi Sandhu. Role and Attribute Based Collaborative Administration of Intra-Tenant Cloud IaaS. In *Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 261–274. ICST, 2014.

[110] Salim Khamadja, Kamel Adi, and Luigi Logrippo. Designing flexible access control models for the cloud. In *Proceedings of the 6th International Conference on Security of Information and Networks - SIN '13*, pages 225–232, New York, New York, USA, nov 2013. ACM Press.

[111] Young-Gab Kim and Yonghan Lee. Context Information-based Application Access Control Model. In *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication - IMCOM '16*, pages 1–5, New York, New York, USA, 2016. ACM Press.

[112] Young-Gab Kim, Chang-Joo Mon, Dongwon Jeong, Jeong-Oog Lee, Chee-Yang Song, and Doo-Kwon Baik. Context-Aware Access Control Mechanism for Ubiquitous Applications. pages 236–242. Springer Berlin Heidelberg, 2005.

[113] S Kirrane, A Mileo, and S Decker. Access control and the resource description framework: A survey. *semantic-web-journal.net*, 2015.

[114] Natsumi Kitajima, Shinsuke Nara, Y Goto, Goto Yuichi, and Jingde Cheng. A Deontic Relevant Logic Approach to Reasoning about Actions in Computing Anticipatory Systems. In Belgium D. M. Dubois, CHAOS, L ege, editor, *International Journal of Computing Anticipatory Systems*, volume 20, pages 177–190. 2008.

[115] Ricky Knights and Emma Morris. Move to intelligence-driven security. *Network Security*, 2015(8):15–18, aug 2015.

[116] Grzegorz Kołaczek. Application of deontic logic in Role-Based Access Control. *International Journal of Applied Mathematics and Computer Science*, 12(2):269–275, 2002.

[117] Thumrongsak Kosiyatrakul. *A modal logic for role-based access control within the HOL theorem prover*. 2010.

[118] Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, and Artsiom Yautsiukhin. Towards Policy Engineering for Attribute-Based Access Control. In *INTRUST 2013: Trusted Systems*, pages 85–102. Springer, Cham, 2013.

[119] Ronald L. Krutz and Russell Dean Vines. *Cloud security : a comprehensive guide to secure cloud computing*. Wiley Pub, 2010.

[120] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, nov 1992.

[121] Neal Leavitt. Is Cloud Computing Really Ready for Prime Time? *Computer*, 42(1):15–20, jan 2009.

[122] Bo Li, Jianxin Li, Lu Liu, and Chao Zhou. Toward a flexible and fine-grained access control framework for infrastructure as a service clouds. *Security and Communication Networks*, pages n/a–n/a, feb 2015.

[123] F. Li. Context-Aware Attribute-Based Techniques for Data Security and Access Control in Mobile Cloud Environment, apr 2015.

[124] Huan-Chung Li, Po-Huei Liang, Jiann-Min Yang, and Shiang-Jiun Chen. Analysis on Cloud-Based Security Vulnerability Assessment. In *2010 IEEE 7th International Conference on E-Business Engineering*, pages 490–494. IEEE, nov 2010.

[125] Jin Li, Gansen Zhao, Xiaofeng Chen, Dongqing Xie, Chunming Rong, Wenjun Li, Lianzhang Tang, and Yong Tang. Fine-Grained Data Access Control Systems with User Accountability in Cloud Computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 89–96. IEEE, nov 2010.

[126] Richard Li, Dallin Abendroth, Xing Lin, Yuankai Guo, Hyun-Wook Baek, Eric Eide, Robert Ricci, and Jacobus Van der Merwe. Potassium: penetration testing as a service. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*, pages 30–42, New York, New York, USA, aug 2015. ACM Press.

[127] Dan Lin, Prathima Rao, Elisa Bertino, Ninghui Li, and Jorge Lobo. Policy decomposition for collaborative access control. In *Proceedings of the 13th ACM symposium on Access control models and technologies - SACMAT '08*, page 103, New York, New York, USA, jun 2008. ACM Press.

[128] Alex X. Liu. Firewall policy change-impact analysis. *ACM Transactions on Internet Technology*, 11(4):1–24, mar 2012.

[129] Meng Liu, Wanchun Dou, Shui Yu, and Zhensheng Zhang. A Decentralized Cloud Firewall Framework with Resources Provisioning Cost Optimization. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):621–631, mar 2015.

[130] Xingang Liu, Jinpeng Huai, Qin Li, and Tianyu Wo. Network state consistency of virtual machine in live migration. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, page 727, New York, New York, USA, mar 2010. ACM Press.

[131] Zhengtao Liu and Jiandong Wang. A fine-grained context-aware access control model for health care and life science linked data. *Multimedia Tools and Applications*, 75(22):14263–14280, jan 2016.

[132] Geethapriya Liyanage and Shantha Fernando. Firewall model for cloud computing. In *2013 IEEE 8th International Conference on Industrial and Information Systems*, pages 86–91. IEEE, dec 2013.

[133] Jun Long, Lu-da Wang, and Zu-de Li. Research and design of the firewall penetration technology serving to distributed cloud resource. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware - Internetware '13*, pages 1–4, New York, New York, USA, oct 2013. ACM Press.

[134] Yang Luo, Wu Luo, Tian Puyang, Qingni Shen, Anbang Ruan, and Zhonghai Wu. OpenStack Security Modules: A Least-Invasive Access Control Framework for the Cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 51–58. IEEE, jun 2016.

[135] Zhongyue Luo. Hacking on OpenStack's Nova source code, 2012.

[136] David MacKenzie. Ubuntu Manpage: time - run programs and summarize system resource usage, 2010.

[137] N Maghanathan. Review of access control models for cloud computing. *Computer Science & Information Science*, 2013.

[138] Rahat Masood and Muhammad Awais Shibli. Comparative Analysis of Access Control Systems on Cloud. In *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 41–46. IEEE, aug 2012.

[139] Fabio Massacci. Reasoning about security: A logic and a decision method for role-based access control. In *Qualitative and Quantitative Practical Reasoning*, pages 421–435. Springer, Berlin, Heidelberg, 1997.

[140] R Mastop. Modal Logic for Artificial Intelligence. 2012.

[141] Violeta Medina and Juan Manuel García. A survey of migration mechanisms of virtual machines. *ACM Computing Surveys*, 46(3):1–33, feb 2014.

[142] Mohamed Mejri and Hamdi Yahyaoui. Formal specification and integration of distributed security policies. *Computer Languages, Systems & Structures*, 49:1–35, sep 2017.

[143] Steve Menard and Luis Nell. JPype documentation JPype 0.6.2 documentation, 2014.

[144] J.-J. Ch. Meyer, R. J. Wieringa, and F. P. M. Dignum. The Role of Deontic Logic in the Specification of Information Systems. In *Logics for Databases and Information Systems*, pages 71–115. Springer US, Boston, MA, 1998.

[145] Natalia Miloslavskaya, Mikhail Senatorov, Alexander Tolstoy, and Sergey Zapechnikov. Big Data Information Security Maintenance. In *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*, pages 89–94, New York, New York, USA, sep 2014. ACM Press.

[146] Chirag N. Modi, Dhiren R. Patel, Avi Patel, and Muttukrishnan Rajarajan. Integrating Signature Apriori based Network Intrusion Detection System (NIDS) in Cloud Computing. *Procedia Technology*, 6:905–912, jan 2012.

[147] M Moshref, M Yu, A Sharma, and R Govindan. vcrib: Virtualized rule management in the cloud. *Proc. NSDI*, 2013.

[148] K Nahrstedt and R Campbell. Security for Cloud Computing. Technical report, Report to the National Science Foundation, 2012.

[149] Michael. Negnevitsky. *Artificial intelligence : a guide to intelligent systems*. Addison Wesley/Pearson, 3rd edition, 2011.

[150] Canh Ngo, Yuri Demchenko, and Cees de Laat. Multi-tenant attribute-based access control for cloud infrastructure services. *Journal of Information Security and Applications*, 27:65–84, dec 2015.

[151] Binh Minh Nguyen, Viet Tran, and L. Hluchy. Abstraction approach for developing and delivering cloud-based services. In *2012 International Conference on Computer Systems and Industrial Informatics (ICCSII)*, pages 1–6, dec 2012.

[152] Dang Nguyen. *Provenance-based access control models*. PhD thesis, The University of Texas at San Antonio, 2014.

[153] Dang Nguyen, Jaehong Park, and Ravi Sandhu. A provenance-based access control model. In *2012 Tenth Annual International Conference on Privacy, Security and Trust*, pages 137–144. IEEE, jul 2012.

[154] Jakob Nielsen. *Usability engineering*. Academic Press, 1993.

[155] Jeremiah Nielsen and Thomas Hacker. Using Virtual Private Networks for Reliable VM Based HPC Systems. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 1226–1233. IEEE, nov 2012.

[156] Nullege. jpype - Nullege Python Samples.

[157] OASIS XACML Technical Committee and others. Extensible access control markup language (XACML) version 3.0. *Oasis standard, OASIS*, 2013.

[158] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Empirical Exploitation of Live Virtual Machine Migration. In *Proceedings of the Black Hat Convention*, 2008.

[159] Ibrahim Yonis Omar, Romain Laborde, Ahmad Samer Wazan, Francois Barrere, and Abdelmalek Benzekri. G-Cloud on Openstack: Adressing access control and regulation requirements. In *2015 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, may 2015.

[160] OpenStack. OpenStack Docs: Configure neutron agents, 2017.

[161] OpenStack. OpenStack Docs: Ocata Installation Tutorials and Guides, 2017.

[162] OpenStack. OpenStack Docs: OpenStack Security Guide, 2017.

[163] OpenStack. OpenStack Docs: policy.json, 2017.

[164] OpenStack Team RedHat. Chapter 1. Components, 2017.

[165] OpenStack.org. Contributors/Corporate - OpenStack, 2017.

[166] OpenStack.org. OpenStack Docs: oslo.policy, 2017.

[167] OpenStack.org. OpenStack Docs: Telemetry service overview, 2017.

[168] OpenStack.org. OpenStack Docs: Welcome to Congress, 2017.

[169] OpenStack.org. Policy in code oslo-specs 0.0.1.dev222 documentation, 2017.

[170] Dustin Owens. Securing elasticity in the cloud. *Communications of the ACM*, 8(5):46, may 2010.

[171] Behzad Pashaie. A Study Evaluating Open Source Cloud Computing Platforms. 2013.

[172] Diego Perez-Botero. A Brief Tutorial on Live Virtual Machine Migration From a Security Perspective. *University of Princeton, USA*, 2011.

[173] Lucian Popa, Minlan Yu, Steven Y. Ko, Sylvia Ratnasamy, and Ion Stoica. CloudPolice. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, pages 1–6, New York, New York, USA, oct 2010. ACM Press.

[174] Paul. Portner. *Modality*. Oxford University Press, 2009.

[175] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle. In *Proceedings of the 2013 conference on Internet measurement conference - IMC '13*, pages 9–22, New York, New York, USA, oct 2013. ACM Press.

[176] N Pustchi and R Sandhu. MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust. In *International Conference on Network and System Security*, pages 206—-220. Springer, 2015.

[177] Himanshu Raj and Karsten Schwan. Extending virtualization services with trust guarantees via behavioral monitoring. In *Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems - VDTS '09*, pages 24–29, New York, New York, USA, mar 2009. ACM, ACM Press.

[178] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. An algebra for fine-grained integration of XACML policies. In *Proceedings of the 14th ACM symposium on Access control models and technologies - SACMAT '09*, page 63, New York, New York, USA, jun 2009. ACM Press.

[179] Khaled Riad, Zhu Yan, Hongxin Hu, and Gail-Joon Ahn. AR-ABAC: A New Attribute Based Access Control Model Supporting Attribute-Rules for Cloud Computing. In *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, pages 28–35. IEEE, oct 2015.

[180] Kenneth H. Rosen. *Discrete mathematics and its applications*. McGraw Hill Higher Education, 7 edition, 2012.

[181] Olivier. Roy, Allard Tamminga, and Malte Willer. *Deontic Logic and Normative Systems. 13th International Conference, Deon 2016, Bayreuth, Germany, July 18-21, 2016*. 2016.

[182] Stuart J. (Stuart Jonathan) Russell, Peter. Norvig, and Ernest. Davis. *Artificial intelligence : a modern approach*. Prentice Hall, 2010.

[183] Sahil, Sandeep Sood, Sandeep Mehmi, and Shikha Dogra. Artificial intelligence for designing user profiling system for cloud computing security: Experiment. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 51–58. IEEE, mar 2015.

[184] Ashok Singh Sairam, Rahul Kumar, and Pratima Biswas. Implementation of an Adaptive Traffic-aware Firewall. In *Proceedings of the 7th International Conference on Security of Information and Networks - SIN '14*, pages 385–391, New York, New York, USA, sep 2014. ACM Press.

[185] Georgia Sakellari and George Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39:92–103, dec 2013.

[186] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *Lecture notes in computer science*, pages 137–196. Springer, Berlin, Heidelberg, 2001.

[187] Ravi Sandhu. The PEI framework for application-centric security. In *Proceedings of the 5th International ICST Conference on Collaborative Computing: Networking, Applications, Worksharing*, pages 1–6. IEEE, 2009.

[188] Ravi Sandhu and Ravi. The authorization leap from rights to attributes. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies - SACMAT '12*, page 69, New York, New York, USA, 2012. ACM Press.

[189] Ravi Sandhu and Pierangela Samarati. Authentication, access control, and audit. *ACM Computing Surveys*, 28(1):241–243, mar 1996.

[190] P Schoo, V Fusenig, V Souza, and M Melo. Challenges for cloud networking security. *Mobile Networks and ...*, 2011.

[191] Cloud secyrity Alliance, Paul Simmonds, Chris Rezek, Archie Reed, and Cloud secyrity Alliance. Security guidance for critical areas of focus in cloud computing v3. 0. *Cloud Security Alliance*, page 176, 2011.

[192] Daniel Servos and Sylvia L. Osborn. HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control. In *Foundations and Practice of Security*, pages 187–204. Springer, Cham, 2015.

[193] Daniel Servos and Sylvia L. Osborn. Current Research and Open Problems in Attribute-Based Access Control. *ACM Computing Surveys*, 49(4):1–45, jan 2017.

[194] Payal Hemchand Shah. Security in live Virtual Machine migration, 2011.

[195] Nitin Kumar Sharma and Anupam Joshi. Representing Attribute Based Access Control Policies in OWL. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 333–336. IEEE, feb 2016.

[196] R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. In *Proceedings 10th Computer Security Foundations Workshop*, pages 183–194. IEEE Comput. Soc. Press, 1997.

[197] A Singh and K Chatterjee. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79:88 —- 115, 2017.

[198] Waleed W. Smari, Patrice Clemente, and Jean-Francois Lalande. An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system. *Future Generation Computer Systems*, 31:147–168, 2014.

[199] A. J. Soroka. Agent-based System for Knowledge Acquisition and Management Within a Networked Enterprise. In *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*, pages 43–86. Springer London, 2010.

[200] Bernard Stepien, Amy Felty, and Stan Matwin. Challenges of Composing XACML Policies. In *2014 Ninth International Conference on Availability, Reliability and Security*, pages 234–241. IEEE, sep 2014.

[201] Norshazrul Azman Bin Sulaiman and Hideo Masuda. Evaluation of a Secure Live Migration of Virtual Machines Using Ipsec Implementation. In *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, pages 687–693. IEEE, aug 2014.

[202] Xin Sun, Xishun Zhao, and Livio Robaldo. Norm-based deontic logic for access control, some computational results. *Future Generation Computer Systems*, 2017.

[203] Yuqiong Sun, Giuseppe Petracca, and Trent Jaeger. Inevitable Failure. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security - CCSW '14*, pages 141–150, New York, New York, USA, nov 2014. ACM Press.

[204] Petter Svärd, Benoit Hudzia, Johan Tordsson, and Erik Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international*

conference on Virtual execution environments - VEE '11, volume 46, page 111, New York, New York, USA, mar 2011. ACM Press.

[205] Takahiro Tagawa and Jingde Cheng. Deontic Relevant Logic: A Strong Relevant Logic Approach to Removing Paradoxes from Deontic Logic. pages 39–48. Springer, Berlin, Heidelberg, 2002.

[206] Bo Tang. *Multi-tenant access control for cloud services*. Phd, The University of Texas at San Antonio, 2014.

[207] Bo Tang and Ravi Sandhu. Extending OpenStack Access Control with Domain Trust. pages 54–69. Springer, Cham, oct 2014.

[208] Zahra Tavakoli, Sebastian Meier, and Alexander Vensmer. A Framework for Security Context Migration in a Firewall Secured Virtual Machine Environment. In Róbert Szabó and Attila Vidács, editors, *Information and Communication Technologies*, Lecture Notes in Computer Science, pages 41–51. Springer Berlin Heidelberg, jan 2012.

[209] Brandon Taylor, Anind K. Dey, Daniel Siewiorek, and Asim Smailagic. Using Crowd Sourcing to Measure the Effects of System Response Delays on User Engagement. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 4413–4422, New York, New York, USA, 2016. ACM Press.

[210] J. Lane Thames, Randal Abler, and David Keeling. A distributed firewall and active response architecture providing preemptive protection. In *Proceedings of the 46th Annual Southeast Regional Conference on XX - ACM-SE 46*, page 220, New York, New York, USA, mar 2008. ACM Press.

[211] Zhu Tianyi, Liu Weidong, and Song Jiaxing. An Efficient Role Based Access Control System for Cloud Computing. In *2011 IEEE 11th International Conference on Computer and Information Technology*, pages 97–102. IEEE, aug 2011.

[212] Adel Nadjaran Toosi, Rodrigo N. Calheiros, and Rajkumar Buyya. Interconnected Cloud Computing Environments. *ACM Computing Surveys*, 47(1):1–47, jul 2014.

[213] Vadim Truksha. Cloud Platform Comparison: CloudStack, Eucalyptus, vCloud Director, and OpenStack. Technical report, Altoros, 2012.

[214] Johan van Benthem. *Modal logic for open minds*. Center for the Study of Language and Information, 2010.

[215] Luis M. Vaquero, Luis Rodero-Merino, and Daniel Morán. Locking the sky: a survey on IaaS cloud security. *Computing*, 91(1):93–118, nov 2010.

[216] MY Vardi. Why is modal logic so robustly decidable? *Descriptive complexity and finite models*, 1996.

[217] Krishna K. Venkatasubramanian, Tridib Mukherjee, and Sandeep K. S. Gupta. CAAC – An Adaptive and Proactive Access Control Approach for Emergencies in Smart Infrastructures. *ACM Transactions on Autonomous and Adaptive Systems*, 8(4):1–18, jan 2014.

[218] Melvin Ver. *Dynamic load balancing based on live migration of virtual machines: Security threats and effects*. PhD thesis, 2011.

[219] Georg Henrik Von Wright. Deontic Logic: A Personal View. *Ratio Juris*, 12(1):26–38, mar 1999.

[220] W. Li, H. Wan, X. Ren, and S. Li. A Refined RBAC Model for Cloud Computing. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 43–48. IEEE, may 2012.

[221] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering - FMSE '04*, page 45, New York, New York, USA, oct 2004. ACM Press.

[222] Lizhe Wang. *Cloud Computing: Methodology, Systems, and Applications*. CRC Press, Boca Raton, 2011.

[223] Wei Wang, Ben Lin, Xiaoxin Wu, and Kai Miao. Secured and reliable VM migration in personal cloud. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 1, pages V1–705–V1–709. IEEE, 2010.

[224] Z Wang, ZH Lu, J Wu, and K Fan. CPFirewall: A Novel Parallel Firewall Scheme for FWaaS in the Cloud Environment. *Advances in Services Computing*, 2015.

[225] Xiaolong Wen, Genqiang Gu, Qingchun Li, Yun Gao, and Xuejie Zhang. Comparison of open-source cloud management platforms: OpenStack and OpenNebula. In *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2457–2461, may 2012.

[226] Jan Wiebelitz, Michael Brenner, Christopher Kunz, and Matthew Smith. Early defense. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, page 336, New York, New York, USA, jun 2010. ACM Press.

[227] R Wu, GJ Ahn, H Hu, and M Singhal. Information flow control in cloud computing. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference*, pages 1–7. IEEE Xplore, 2010.

[228] Kun Xu, Chuang Lin, Zhen Chen, Kun Meng, and Mourad Hakmaoui. An Effective Policy Relocation Scheme for VM Migration in Software-Defined Networks. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, aug 2015.

[229] Xuanxia Yao, Xiaoguang Han, and Xiaojiang Du. A lightweight access control mechanism for mobile cloud computing. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 380–385. IEEE, apr 2014.

[230] Adam Young. Dynamic Policy for Access Control, 2015.

[231] Younis A. Younis, Kashif Kifayat, and Madjid Merabti. A novel evaluation criteria to cloud based access control models. In *2015 11th International Conference on Innovations in Information Technology (IIT)*, pages 68–73. IEEE, nov 2015.

[232] Shui Yu, Robin Doss, Wanlei Zhou, and Song Guo. A general cloud firewall framework with dynamic resource allocation. In *2013 IEEE International Conference on Communications (ICC)*, pages 1941–1945. IEEE, jun 2013.

[233] Vladimir Zaborovsky, Alexey Lukashin, Sergey Kupreenko, and Vladimir Mulukha. Dynamic access control in cloud services. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1400–1404. IEEE, oct 2011.

[234] Edward N Zalta. Basic Concepts in Modal Logic. *Center for the Study of Language and Information Publications*, page 92, 1988.

[235] Fengzhe Zhang, Yijian Huang, Huihong Wang, Haibo Chen, and Binyu Zang. PALM: Security Preserving VM Live Migration for Systems with VMM-enforced Protection. pages 9–18. IEEE, oct 2008.

[236] Ni Zhang, Di Liu, and Yunyong Zhang. A Research on Cloud Computing Security. In *2013 International Conference on Information Technology and Applications*, pages 370–373. IEEE, nov 2013.

[237] Su Zhang, Xinwen Zhang, and Xinming Ou. After we knew it. In *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, pages 317–328, New York, New York, USA, jun 2014. ACM Press.

[238] Yun Zhang, Ram Krishnan, and Ravi Sandhu. Secure Information and Resource Sharing in Cloud Infrastructure as a Service. In *Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security - WISCS '14*, pages 81–90, New York, New York, USA, nov 2014. ACM Press.

[239] Yun Zhang, Farhan Patwa, Ravi Sandhu, and Bo Tang. Hierarchical Secure Information and Resource Sharing in OpenStack Community Cloud. In *2015 IEEE International Conference on Information Reuse and Integration*, pages 419–426. IEEE, aug 2015.

[240] Chao Zhou and Bo Li. iHAC: A Hybrid Access Control Framework for IaaS Clouds. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 853–858. IEEE, dec 2014.

[241] H Zhou, J Wang, and HG Zhang. A Trusted VM-vTPM Live Migration Protocol in Clouds. 2013.

[242] Yan Zhu, Dijiang Huang, Chang-Jyun Hu, and Xin Wang. From RBAC to ABAC: Constructing Flexible Data Access Control for Cloud Storage Services. *IEEE Transactions on Services Computing*, 8(4):601–616, jul 2015.