

KI-NGĀ-KŌPUKU: A DECENTRALISED, DISTRIBUTED SECURITY MODEL FOR CLOUD COMPUTING

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisors

Alan T Litchfield

Brian Cusack

August 2018

By

Monjur Ahmed

School of Engineering, Computer and Mathematical Sciences

Abstract

The research proposes a decentralised and distributed security model for Cloud Computing in the form of a development framework. The distributed nature of Cloud infrastructure makes it a very complex one. This research examines whether the distributed nature of Cloud resources is a contributing factor for secured Cloud Computing. The research also seeks answer to whether a decentralised and distributed approach for the distributed Cloud resources is more credible than a centralised approach. The proposed security model is named Ki-Ngā-Kōpuku ¹.

To present Ki-Ngā-Kōpuku, the concept and definition of Cloud Computing is explored. State-of-the-art Cloud Computing security and security models have also been explored, with a specific focus on finding the worthiness of conducting research for a decentralised and distributed Cloud security model. The focus is also on finding Cloud security models that are distributed in nature. Design Science Research is determined as the research methodology for the conducted research, which is aided by Formal Methods for validation. Formulation of research questions and hypotheses as well as approaches to test the hypotheses are addressed in discussing the methodology. An adapted Design Science Research framework is used with three main stages: Problem Identification, Solution Design and Evaluation. In Problem Identification, literature review leads to formulate research questions that leads to design the artefacts in Solution

¹The word Ki-Ngā-Kōpuku is from the Māori language and translates to "at the nodes." It is pronounced key-ngar-kaw-puhkuh.

Design stage. The Evaluation stage of Design Science Research then helps to validate the design artefacts where formal methods are used as validation tool within the adapted Design Science Research framework.

Ki-Ngā-Kōpuku outlines a security model in the form of a software development framework. It consists of a reference architecture and an associated security mechanism. It provides a means to secure an application such that an application cannot be taken down resulting in service unavailability. Ki-Ngā-Kōpuku does so by dividing an application into several parts and randomly distributing the parts into random Cloud servers, thus making the application distributed as well as decentralised. The distinct features of Ki-Ngā-Kōpuku are being distributed and decentralised by means of redundancy that results in having no single point of failure, and makes it a self-healing system.

The problem is then defined and the specifications of the proposed security model are outlined. Software Requirements Specifications and the framework perspective Software Requirements Specifications for Ki-Ngā-Kōpuku form part of the problem analysis. The system architecture based on the specification is then developed followed by discussion on the associated security mechanism.

The proof of concept is done through logical validation and logical simulation. Validation of various aspects of the proposed security model is done by logical modelling. Formal methods and logical reasoning are used to establish the logical validation of the system. Finally, the analysis and discussion of the research findings are noted. Possible future developments and enhancement of the proposed security model forms major part of the concluding discussion.

Contents

Abstract	2
Attestation of Authorship	11
Publications	12
Acknowledgements	13
Dedication	14
1 Introduction	15
1.1 Introduction	15
1.2 Background	17
1.3 Research Aim	19
1.4 Research Significance and Novelty	20
1.5 Ki-Ngā-Kōpuku in a Nutshell	21
1.6 Thesis Structure	22
1.7 Conclusion	25
2 Literature Review	26
2.1 Introduction	26
2.2 CC Concept	29
2.2.1 Cloud Types	30
2.2.2 Cloud Service Architecture	32
2.2.3 Cloud and Distributed Computing	33
2.2.4 Cloud and Virtualisation	34
2.3 Security and CC	35
2.3.1 Degree to which Threats and Vulnerabilities affect Cloud	35
2.3.2 Importance of Security in Cloud	37
2.4 Security Points in Cloud Architecture	38
2.4.1 Known Threats and Vulnerabilities in Computing	38
2.4.2 Cloud-Specific Threats and Vulnerabilities	40
2.4.3 Cloud Threat Taxonomy	44
2.4.4 Architectural Weakness for Vulnerabilities	63
2.5 Security Models for CC	64

2.5.1	Concept of Security Model in Cloud	64
2.5.2	Security Models and Mechanisms for Cloud	65
2.5.3	Existing Security Models at a glance	69
2.6	Distributing Security Model: Implications	73
2.7	Implications of Redundancy in CC	76
2.8	Management Frameworks	77
2.9	Research Questions (RQs)	78
2.10	Conclusion	79
3	Research Methodology	80
3.1	Introduction	80
3.2	Methodology and Research Process	81
3.2.1	Methodology	82
3.2.2	Research Process	82
3.2.3	DSR Evaluation	85
3.2.4	Other Research Methods	86
3.3	Research Questions (RQs) and Hypotheses	88
3.3.1	Research Questions	88
3.3.2	Hypotheses & Testing Methods	89
3.4	Conclusion	95
4	Problem Analysis	96
4.1	Introduction	96
4.2	Problem Definition	97
4.2.1	Design Definition	97
4.3	Design Specification	98
4.3.1	Software Requirements Specification	99
4.3.2	Framework Perspective SRS	108
4.4	Types of Security Provided	109
4.5	Conclusion	110
5	System Architecture	112
5.1	Introduction	112
5.2	Distributed Security Model	113
5.2.1	Security Model and Security Mechanism	113
5.2.2	The Model	114
5.3	Reference Architecture and Security Mechanism	119
5.4	Ki-Ngā-Kōpuku Life Cycle	122
5.5	Conclusion	124
6	Processing within Ki-Ngā-Kōpuku	125
6.1	Introduction	125
6.1.1	Assumptions	126
6.2	Security Mechanism	127

6.2.1	Componentisation	127
6.2.2	Elements of a Node	128
6.2.3	Add New Node	131
6.2.4	Inter Component Interfacing	136
6.2.5	Incident Monitoring and Response	142
6.3	Conclusion	143
7	Proof of Concept	144
7.1	Introduction	144
7.2	Validation	145
7.2.1	Model Validation	146
7.2.2	Turing Machine	149
7.2.3	Security Mechanism Validation	151
7.2.4	Logical Simulation of Distribution	154
7.3	Further Validation	160
7.3.1	WannaCry	160
7.3.2	Mirai Botnets	161
7.4	Conclusion	161
8	Discussion	163
8.1	Introduction	163
8.2	Research and Research Design: Mapping	164
8.2.1	Research Question 1	164
8.2.2	Research Question 2	167
8.2.3	Research Question 3	170
8.2.4	Reflection on Research Achievements	178
8.3	Complementing other Technologies	181
8.3.1	Blockchain and Ki-Ngā-Kōpuku	181
8.3.2	Machine Learning	182
8.3.3	Cryptography	183
8.4	Conclusion	183
9	Conclusion	185
9.1	Introduction	185
9.2	Future Work	188
9.2.1	Architecture Dependency	189
9.2.2	Automation of Randomisation	189
9.2.3	Data Security	190
9.2.4	Incident Monitoring	190
9.2.5	Existing Performance Bottleneck	191
9.3	Concluding Remarks	191
	References	193

List of Tables

2.1	Summary of Attacks in terms of the Taxonomy	47
2.2	Threats in Literature Categorized according to the Taxonomy	48
2.3	Summary of the Security Models or Mechanisms	71
7.1	Instruction Set for Turing Machine	150
7.2	Initial Distribution of an Application with Three Components	158
7.3	Changed Distribution Scenario with no Decrement in Component Count	159
8.1	Evidence of Research Achievements	174


List of Figures

1.1	Structure of the Thesis	23
2.1	CC Schematic Definition	30
2.2	NIST definition of the Cloud framework	31
2.3	Threat Taxonomy for CC	56
3.1	Research Process	83
3.2	RQ, hypothesis, and method mapping	95
4.1	Problem Specification	99
4.2	Frozen and Hot Spot	108
5.1	Contextual View of the Distributed Security Model	119
5.2	Reference Architecture for Ki-Ngā-Kōpuku	120
5.3	Contextual View of the Security Mechanism	122
5.4	Life cycle for Ki-Ngā-Kōpuku	123
5.5	Logical View of Ki-Ngā-Kōpuku	123
6.1	Elements of a Node	128
6.2	Transformation of a Default Node into Application Node	130
6.3	Add New Node	132
6.4	Component Distribution	133
6.5	Component Selection	135
6.6	Function Calls among Components	138
6.7	Inter-Component Interfacing	140
6.8	Target Coupling	141
7.1	NFA Representation of the Distributed Security Model	146
7.2	DFA Representation of the Distributed Security Model	148
7.3	Turing Machine representation of the distributed security model	151
7.4	Proof of Concept of the Security Mechanism	153
7.5	Component Count Database	156
8.1	Mapping RQ-1, Hypothesis and Testing Methods	164
8.2	Mapping RQ-2, Hypotheses and Testing Methods	167
8.3	Mapping RQ-3, Hypothesis and Testing Methods	171

B.1	Elements of the Originating BEC	215
B.2	Self-destruction Mechanism for Originating BEC	216
B.3	Add New CIOS	218
B.4	Component Distribution	220
B.5	Component Selection	222
B.6	Key Creation and Distribution	224
B.7	Data Transfer among BECs	225
B.8	The Chosen One	228
B.9	The Voting Process	229
B.10	Incident Monitoring	231
C.1	Computing Architecture for CORP	233
C.2	CORP Algorithm	236
C.3	Collaborative Processing using CORP	237

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

A handwritten signature in black ink, reading "Hongjiu Ahmed." with a period at the end. The script is cursive and fluid.

Signature of student

Publications

A Distributed Security Model for Cloud Computing

22nd Americas Conference on Information Systems (AMCIS 2016), San Diego, USA.

Taxonomy for Identification of Security Issues in Cloud Computing Environments

Journal of Computer Information Systems (JCIS), Taylor & Francis, 2016.

A Generalized Threat Taxonomy for Cloud Computing

25th Australasian Conference on Information Systems (ACIS 2014), 8-10 December, Auckland, New Zealand.

Acknowledgements

It was a good journey throughout my doctoral research period. I was lucky to have Dr Alan T Litchfield as my Primary Supervisor. His supervision and guidelines would surely have a lifelong positive impact on me - to learn, to grow, and to succeed. As an AUT student, I am one of those who have benefited from using his developed L^AT_EXthesis template that I have used to write this thesis. I would also like to thank my Secondary Supervisor Dr Brian Cusack for his guidelines and suggestions from time to time. Thanks to all my colleagues and friends at Service and Cloud Computing Research Lab (SCCRL). Special thanks to Chandan for his help with formal methods, to Karishm for being fabulously helpful throughout the PhD course, and to Bumjun Kim for prompt support with network infrastructure. I would also like to thank Abid, Joyce, Naga and Nurul for their support during my research period.

Dedication

Dad,

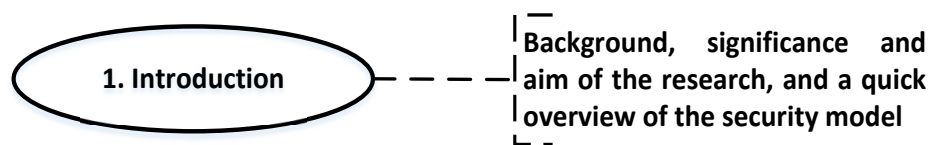
I can only imagine how happy you are to see this.

Supaporn Jaysakul,

I always feel privileged to have met a person like you.

Chapter 1

Introduction



1.1 Introduction

The research proposes a decentralised and distributed security model for Cloud Computing (CC) in the form of a development architecture. The distributed nature of Cloud infrastructure makes its management a very complex one (Kumar, Nitin, Sehgal, Shah & Chauhan, 2011). The research examines the implications of the distributed nature of Cloud resources towards security in Cloud Computing. The research also seeks answer to whether a decentralised and distributed approach for the distributed Cloud resources provide more security than the centralised one. Both primary and secondary research is conducted by following Design Science Research (DSR) as the research methodology, and Formal Methods are used as an evaluation and validation tool within DSR. The proposed security model is named Ki-Ngā-Kōpuku. Ki-Ngā-Kōpuku outlines a security

model in the form of a software development architecture. It consists of a Reference Architecture (RA) and an associated security mechanism. It provides a means to secure an application such that an application cannot be taken down that may result in service unavailability. Ki-Ngā-Kōpuku does so by dividing an application into several parts and then randomly distributing the parts into random Cloud servers, thus making the application distributed as well as decentralised. The distinct features of Ki-Ngā-Kōpuku are being distributed and decentralised by means of redundancy that results in having no single point of failure, and makes it a self-healing system. As the proposed security model is for CC, the concepts and features of CC are explored as background research for Ki-Ngā-Kōpuku.

CC is a contemporary computing approach. In recent years, the computing environment has experienced a significant transition to CC. To retain its integrity and to establish its acceptance, CC needs to satisfy the target audience that it is a safer computing approach. As a result, security is a prioritised concern for CC. The thesis presents Ki-Ngā-Kōpuku, a decentralised and distributed security model for CC. As part of its security strategy, Ki-Ngā-Kōpuku distributes an application by dividing it into several parts or components, and then scattering the components into different computers or Cloud servers. Such componentised distribution makes Ki-Ngā-Kōpuku decentralised, as there is no ‘core’ and central management of the system. Thus, Ki-Ngā-Kōpuku is a decentralised and distributed security model for CC.

The context and the big picture of the conducted research on Ki-Ngā-Kōpuku are presented in this introductory chapter. The background to the research is discussed in Section 1.2 on the following page. The aim of the research is discussed in the next section, followed by Section 1.4 on page 20 that notes the significance of the research. Section 1.5 on page 21 consists of a quick overview of Ki-Ngā-Kōpuku as a security model, and the structure of the thesis is illustrated in Section 1.6 on page 22.

1.2 Background

CC means using remote computing resources without worrying about the management of those remote resources (Casola, Cuomo, Rak & Villano, 2013). There is no standard single definition for CC. Relevant literature suggest that CC is a computing approach where required resources are situated at a remote location and the access to them is normally through the Internet (Ryan & Falvey, 2012; Khorshed, Ali & Wasimi, 2012; Jorissen, Vila & Rehr, 2012).

For the purpose of this study, a definition of CC is provided as follows:

CC is a conceptual computing approach that may encapsulate any other computing means and act as a wrapper for all kinds of computing practices. CC is the setting where hard (e.g. network infrastructure) and soft (e.g. data, software, processing) elements are remotely existent and access to these resources is on an ad-hoc basis using public or private communication infrastructure, where the management and maintenance concerns of the Cloud infrastructure including the resources held within the infrastructure are most often beyond the end-users' scope.

CC helps people to use computing resources as and when required, without owning the resources and without worrying about their management. This on-demand only usage of resources significantly reduces costs (Dukaric & Juric, 2013) related to Information Technology (IT) infrastructure deployment – a major reason for CC to have its popularity risen exponentially over the period of time. As a result, new business entities like Cloud Service Providers (CSPs) are introduced, who provide their Cloud infrastructure and related Cloud services to end-users. A CSP's infrastructure thus may hold information from all of its customers, for example different businesses and individuals. In this way, the Cloud infrastructure becomes goldmine of information that may include sensitive information. Cloud infrastructures are lucrative targets for attackers (A. Patel, Taghavi, Bakhtiyari & Junior, 2013) due to being such goldmines,

and having other reasons; which subsequently raises security concerns for CC. CC offers numerous benefits (Akande, April & Belle, 2013), and at the same time, raises security concerns (Jaeger, Lin & Grimes, 2008) for the Cloud infrastructure as well as the resources held within those infrastructure.

The growth of CC has brought a number of advantages (A. Lin & Chen, 2012) and new business perspectives are emerging (Zonouz, Houmansadr, Berthier, Borisov & Sanders, 2013; Svantesson & Clarke, 2010). CC benefits both customers and businesses with its flexibility but at the same time, security concerns have been at the core of discussions (Zissis & Lekkass, 2012).

Data storage and processing in Cloud services are carried out either on the data owner's or through someone else's server and IT infrastructure (Svantesson & Clarke, 2010; Lombardi & Pietro, 2011). Cloud services are subject to attack. Information normally traverses public communication channels and this is a concern because data can be intercepted by unauthorised third parties. The trustworthiness of CSPs is also of concern (Zissis & Lekkass, 2012). Additionally, possible hacking of Cloud infrastructure cannot be avoided (Ryan & Falvey, 2012). The target may be situated beyond an organisation's firewall or in a different country (Svantesson & Clarke, 2010). Privacy, integrity and confidentiality of data are issues in CC. When access to the Cloud is via smart-phones, the security of the smart phones as well as wireless communications also enter into picture (Zonouz et al., 2013). The increased complexity of Cloud infrastructure may create opportunity for attack (King & Raja, 2012).

Virtualisation technology is widely used in CC. The security concerns of virtualisation technology are further issues for CC. A number of security mechanisms exist for CC (Lombardi & Pietro, 2011), for example, the data storage security model (H. B. Patel, Patel, Borisaniya & Patel, 2012), virtualisation security framework (Park, 2012) and trust evaluation model for CC (Wu, Zhanga, Zeng & Zhoua, 2013). But arguments exist that CC requires a new security model to retain its integrity (Gritzalis, Mitchell,

Thuraisingham & Zhou, 2014a). As a result, innovative approaches to a security model for CC are required.

CC adapts a distributed approach to its resource and service provisioning. That is, in the Cloud, the resources and data may be distributed among different servers or even different Cloud infrastructure. The resource distribution in the Cloud makes it complex to manage and administer Cloud infrastructure; which at the same time, makes the Cloud an alluring target for attackers. Consensus exists among researchers that security is one of the major concerns for CC. It is also stated that the current state of Cloud security is not well defined and the adequacy of current Cloud security is not satisfactory (Fernandes, Soares, Gomes, Freire & Inacio, 2014). Due to the distributed nature of CC, complexity of Cloud management as well as security being a sensitive driving factor for such a distributed environment. Arguments exist that a distributed approach towards Cloud security may be a better solution to address Cloud security concerns, since the Cloud itself and its resources are distributed in nature (Poh et al., 2013; Yan, Zhang, Chen, Zhao & Li, 2011).

1.3 Research Aim

The research aim is to explore whether a decentralised and distributed approach to Cloud security is better suited compared to its centralised counterpart in order to eliminate single point of failure, introduce redundancy to subsequently enhance service availability and to achieve self-healing capability of Cloud applications. Though the security model is developed for security, it is adaptable to other applications.

Existing approaches to security are not immune to various attacks, for example Distributed Denial of Service (DDoS) attack. This results for systems to suffer from reliability and availability issues. Data and resources may reside in different premises in a CC scenario. As a result, distributed resources and their management aspects tend

to be complex, which is a reason for CC to be of interest to attackers. It can be argued that a distributed computing scenario requires a distributed approach to address its security concerns - as discussed in Chapter 2 on page 26 that CC is distributed but a truly distributed security model for CC does not exist to date. This acts as the motivation to set the aim to develop Ki-Ngā-Kōpuku as a distributed security model for CC.

The security model is envisioned to have no single point of failure so that compromising one specific resource does not result in service unavailability. To achieve this, the security model needs to be decentralised in such a way that there will be no central core of the system to eliminate a single point of failure. Another major feature of the security model is to be self-healing. That is, if part of the system is breached or compromised, the system itself is able to recover and ‘heal’ its ‘wounded’ part.

1.4 Research Significance and Novelty

The outcome of the research is the specifications of a decentralised and distributed security model for CC. The research contributes both to the related research and industry domain.

The research outcome helps to understand state-of-the-art CC security. It in turn helps to structure Cloud threats and to develop a contemporary Cloud threat taxonomy. The threat taxonomy helps in further research and development in computer, information and Cloud security. Understanding Cloud threats in the form of taxonomy will also help security managers and engineers to better develop disaster recovery plan and information security management plan.

It is found that a decentralised and distributed approach is better suited for Cloud security, though a truly decentralised and distributed security model does not exist to date to the best of the author’s knowledge. Also, a proper and standard definition for security model is not found. This, along with the purely decentralised and distributed

framework based approach towards security are what added to the body of knowledge by the presented research. This is where the novelty of the research stands.

CSPs and Cloud application developers can deploy secure applications and services using the Ki-Ngā-Kōpuku architecture. Thus, Ki-Ngā-Kōpuku may potentially contribute to the area of service-oriented computing. Applications and standard blue-prints of application, for example, accounting software, Enterprise Resource Planning (ERP) systems and other cross-functional and cooperative systems and Information Systems (IS) can be developed as off-the-shelf solutions. As a result, Ki-Ngā-Kōpuku creates an opportunity for software vendors to offer a secured, decentralised, distributed Commercial Off-the-Shelf (COTS) solution.

As it is deemed, and as apparent through the literature review presented in Chapter 2, a purely distributed and decentralised security model like Ki-Ngā-Kōpuku does not exist to date.

1.5 Ki-Ngā-Kōpuku in a Nutshell

Ki-Ngā-Kōpuku is a security model for CC. Ki-Ngā-Kōpuku is not a conventional approach to secured computing. It provides an architecture for application development. Using this architecture to develop an application makes it secure in terms of availability and context illiteracy. The term 'context illiteracy' can be defined as having no capability for third parties to understand the overall context or architecture of a system. As part of its security strategy, Ki-Ngā-Kōpuku makes an application distributed by dividing it into several parts or components, and then randomly (i.e. not following any specific and pre-defined pattern to avoid predictability) scattering the components into different computers or Cloud servers. The scattered and componentised distribution creates a scenario where an attacker is not able to collect the big picture of the application. This exhibits the context illiteracy feature for an application developed using Ki-Ngā-Kōpuku.

Also, the componentised distribution results in having no ‘core’ or central management of the system. This makes Ki-Ngā-Kōpuku decentralised. Thus, Ki-Ngā-Kōpuku is a decentralised and distributed security model for CC.

The system consists of nodes that hold the components of an application. All the nodes collectively defines the scope of the system. No single node is inferior or superior to any other node in terms of capability or priority. Ki-Ngā-Kōpuku is a system of systems made up of nodes and all the nodes collectively are the ‘heart’ of the system. Hence, the name is chosen for the proposed security model.

1.6 Thesis Structure

Figure 1.1 on the next page illustrates the thesis structure. The research on Ki-Ngā-Kōpuku is presented in the thesis through 10 Chapters. In Chapter 1 on page 15, the background, research aims and significance are discussed. A quick overview of the security model is also presented.

The findings in existing literature are discussed in Chapter 2 on page 26. The concept of CC is explored to define CC. Terms such as virtualisation and, distributed computing that are often associated and mentioned in a CC context are also discussed. The preliminary discussion in this Chapter addresses different Cloud deployment models. Finding existing security threats and the sensitivity of CC towards vulnerability are explored through the literature review. The literature review forms the basis to analyse the implications of security in the Cloud and the implications of resource distribution and subsequently to develop a distributed security model for CC. Existing Cloud security models are discussed in this Chapter. Since Ki-Ngā-Kōpuku is a distributed security model, the specific focus is on whether there exists any distributed security model for CC, and if so, to what extent the security model is distributed. The findings through the literature review resulted in establishing the research motivation and the state-of-the-art

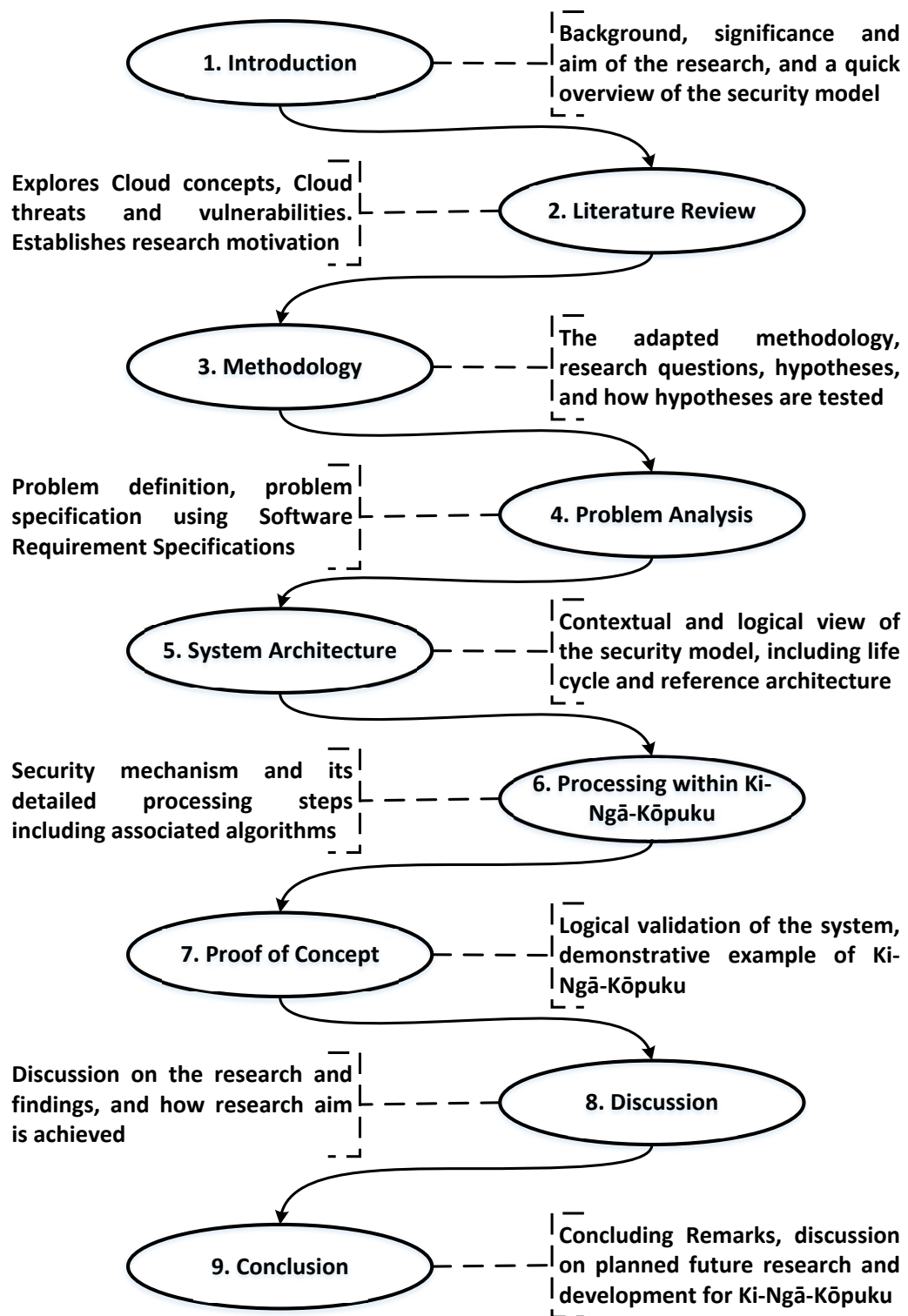


Figure 1.1: Structure of the Thesis

of research in Cloud security. A number of recent Cloud breaches are analysed which, in combination with other literatures, helps to develop a Cloud threat taxonomy. The developed taxonomy based on the literature review is one of the major milestones in the research. Literature review also lead to formulate the Research Questions (RQs).

Chapter 3 on page 80 discusses the methodology and the research design. The RQs are revisited and formulated hypotheses are presented in this Chapter. Additionally, the methods by which the hypotheses are tested are also discussed. A mapping among the RQs, hypotheses and the hypotheses-testing methods is also illustrated and discussed. The research is carried out using DSR as the research methodology where Formal Method is used as the DSR artefacts evaluation and validation tool. The core research methods and the framework to establish the research process are part of the discussion in this Chapter.

Chapter 4 on page 96 is Problem Analysis. It addresses both the problem definition and problem specification. In the problem definition, the research problem that the research aims to solve is discussed. After outlining the problem, further specifics to the proposed security model are outlined in problem specification. A Software Requirements Specifications (SRS) model is used to specify the high-level specifications of the developed security model. The presented SRS acts as the baseline document for the design and development of the security model. Since Ki-Ngā-Kōpuku is an architecture for secure CC, the framework-perspective SRS is also part of Section 4.3 on page 98 in Chapter 4 on page 96.

In later discussions, it is argued that a security model needs to consist of a Reference Architecture (RA) and an associated security mechanism. Thus, based on the specification discussed in the Problem Analysis Chapter, the architecture of Ki-Ngā-Kōpuku is presented in Chapter 5 on page 112. The term Security Model is defined. The contextual and logical high-level view of Ki-Ngā-Kōpuku is described in this Chapter. A bird's eye view of the associated security mechanism is also outlined, followed by the life-cycle

and the RA for Ki-Ngā-Kōpuku.

The security mechanism for Ki-Ngā-Kōpuku details the processing steps and associated algorithm to complete the picture of the security model. The processing approaches for Ki-Ngā-Kōpuku are considered in Chapter 6 on page 125. Processing steps described in this Chapter illustrate how Ki-Ngā-Kōpuku is a decentralised and distributed security model. The explanation also illustrates the features that make Ki-Ngā-Kōpuku a self-healing system, and a system of systems with no single point of failure.

Chapter 7 on page 144 portrays the Proof of Concept for the security model discussed in two earlier Chapters (i.e. Chapters 5 on page 112 and 6 on page 125). The proof of the concept for Ki-Ngā-Kōpuku is done using two approaches – by means of logical validation, logical simulation, and by demonstrating a basic illustrative example. On validating the system, logical and mathematical reasoning are used to establish validity of the security model and its underlying processing approaches and algorithms.

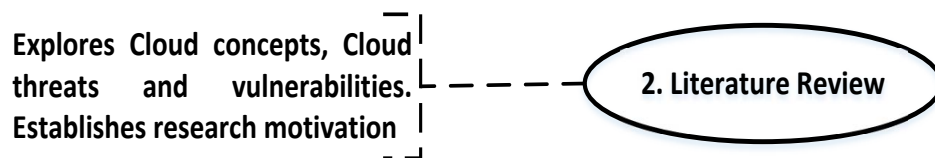
Analysis of the findings as well as the conducted research on Ki-Ngā-Kōpuku are discussed in Chapter 8 on page 163. The envisioned future enhancement and planned future development are outlined in Chapter 9 on page 185 which also includes the concluding remarks, since this is the final Chapter of the thesis.

1.7 Conclusion

The significance of new security models for CC is warranted by further and ongoing research on CC security. In this regard, it is necessary to understand the concept of CC and the security threats that exist in CC. The implications of security for CC and the importance of research investigation on a distributed security model also need appreciation. Such appreciation is realised through the existing literature review. Next Chapter presents the literature review where the above aspects related to CC are explored.

Chapter 2

Literature Review



2.1 Introduction

In this chapter, the findings from relevant literature is presented. The concept of CC as well as the approaches to Cloud deployment including Cloud service architecture are explored. The relation of CC to distributed computing and virtualisation is reviewed. Various security aspects and the state-of-the-art Cloud security forms one of the major parts of the discussion. While exploring security aspects for CC, the implications of security in CC are taken into consideration. The vulnerabilities in computing are explored along with threats and vulnerabilities specific to CC. The findings lead to the proposition of a threat taxonomy for CC. Finally, existing security models for CC are explored to reveal whether distributed security models for CC exist or not; and if so, to which extent.

CC has recently gained popularity and becoming a major contemporary computing approach. It is an innovative approach to computing that incorporates other computing means and tactics. Recently CC has been accepted generally as a means to deploy various computing resources (Dahbur, Mohammad & Tarakji, 2011). CC offers significant cost reductions by using highly scalable hardware and software (Dukaric & Juric, 2013) that are driving factors behind its popularity. Though CC relieves a number of burdens and brings flexibility to its users (Akanke et al., 2013), security is a major concern (Jaeger et al., 2008) and good security measures are considered to be a key requirement for Cloud deployment (Gonzalez et al., 2012). Since CC is being deployed on a massive scale, the security concerns are proportionately of interest.

In CC, the benefits come with shortcomings, particularly from the perspective of security threats and vulnerabilities. For example, the Apple iCloud Breach (Apple, 2014; PCWorld, 2014; TechTimes, 2014), attack on Sony through Amazon Cloud (Bloomberg, 2011a), JP Morgan Cloud server hack (CNN, 2014; Computerworld, 2014), Dropbox password breach (CNN, 2011), SnapChat Cloud servers hack (TheRegister.co.uk, 2014) and service shut-down of New Zealand's largest Telecom vendor, Spark (Stuff.co.nz, 2014). IdentityForce (2017) mentions 29 significant recent Cloud breaches. For example InterContinental Hotels (Osborne, 2017), E-Sports Entertainment Association (ESEA) (Ragan, 2017), Dun & Bradstreet (FoxNewsTech, 2017), and OneLogin (BBC, 2017).

Access to service via public infrastructure is a reason why network security challenges are a common problem (Yang, Wang, Yu, Liu & Peng, 2012). A Cloud-based infrastructure is a distributed network to provide operations. Any distributed network is prone to security issues where the failure of any single element may expose the whole network to attack (Garcia-Morchon, Kuptsov, Gurtov & Wehrle, 2013). CC has a distributed and open structure that makes it a prime target for cyber-attacks (A. Patel et al., 2013). The current state of security measures for CC is puzzling for some researchers (Fernandes et al., 2014) while others suggest that new security mechanisms

are required to sustain its acceptability (Gritzalis, Mitchell, Thuraisingham & Zhou, 2014b). Such opinions suggest that special attention is required for CC security and the security challenges need to be defined and identified.

This chapter presents secondary research on CC and CC security through literature review. The concept of CC including its deployment models is discussed in Section 2.2 on the next page. It also addresses how CC relates to distributed computing, and the scope of virtualisation within the Cloud context. In Section 2.3 on page 35, the significance and importance of security for CC is taken into account. Two major aspects are of interest in this section: the degree to which Cloud is affected by threats and the degree to which CC requires security measures. In Section 2.4 on page 38, the discussion focuses on the known vulnerabilities in computing and then on the Cloud-specific vulnerabilities to map an understanding of how known security issues in other computing approaches are applicable to the CC context. The discussion in Section 2.4 on page 38 portrays the known security threats to CC. Existing security models in CC are discussed in Section 2.5 on page 64; it helps to gain an understanding of the state-of-the-art Cloud security models. The results of a systematic literature review are presented in Section 2.5 on page 64. In this section, the distinction between a security model and a security mechanism for CC is addressed. Once the findings on Cloud security models and whether the discussed models are distributed or not are established, concentration is focused on the implications and importance of the distributed security model for CC in Section 2.6 on page 73. The implications of redundancy in CC is described in Section 2.7 on page 76. Finally, the RQs are developed based on findings in existing literature, and presented in Section 2.9 on page 78.

2.2 CC Concept

This section presents the concept and practice of CC. The defining factors of the term ‘CC’ are explored, different types of Cloud and the service deployment architecture of CC are also of concern in this section. And the significance and relationship of CC to virtualisation as well as distributed computing are presented.

CC is a means to access remote resources generally through a public infrastructure (e.g. Internet) (Ryan & Falvey, 2012). Due to its distributed nature, a Cloud architecture can be geographically dispersed or can be a distributed architecture in a more confined location (Mackay, Baker & Al-Yasiri, 2012), for example in a building or campus.

A formal definition of CC outlined by National Institute of Standards and Technology (NIST) is described by Casola, Cuomo, Rak and Villano (2013) where CC is termed as a convenient way for computing and networking resources to be accessed on an on-demand basis with minimal end-user effort, and with service providers’ minimum interaction. Alongside this formal definition, it is found that the definition is used with different concepts (Arshad, Townend & Xu, 2013). CC is described with different terminologies for example ‘pay as you go computing’, ‘utility computing’ and ‘on demand computing’ (Fernando, Loke & Rahayu, 2013). Jorissen, Vila and Rehr (2012) confine Cloud Computing to being a service accessible over the Internet by means of virtualised and dynamically scalable resources.

Petcu, Macariu, Pania and Craciun (2013) state that CC is an architecture that is capable of providing programmable computing infrastructure. Zissis and Lekkas (2012) describe CC as an innovative Information Systems (IS) architecture with the note that it is envisioned to be future-focused. Additionally, they state that the deployment of CC may change the traditional perception of computing and Operating Systems (OSs), and may lower the level of computing complexity to end-users. In the CC model, users do not need to know where the resources of the service are located or how they are

maintained or administered (Buyyaa, Yea, Venugopala, Broberg & Brandic, 2009).

Figure 2.1 is a schematic illustration of CC (Khorshed et al., 2012):

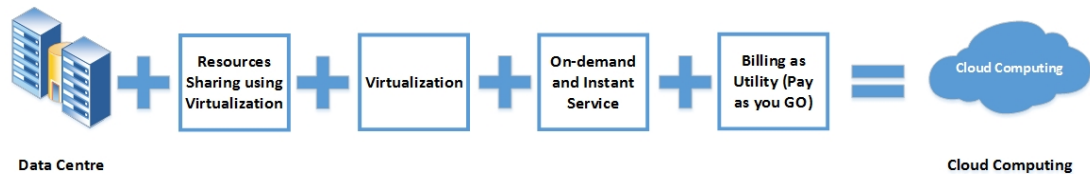


Figure 2.1: CC Schematic Definition (Khorshed et al., 2012)

Figure 2.1 shows that CC architecture comprises at least one data centre using virtualisation technology and resource sharing by means of virtualisation. It is an on-demand service and uses a utility-style billing model. Thus, remote resource utilization is at the core of CC. The Cloud architecture is accessed via public infrastructure therefore some kind of security mechanism is required to ensure safe data transfer. Data centres have an array of servers and other resources that may be dispersed across different sites or geographic locations. Therefore, a Cloud architecture is purely distributed in nature. Additionally, operations and techniques for cluster and grid computing are applicable to the functioning and service delivery of CC. Addressed in next the section, Cloud deployment is associated with different deployment and service models.

2.2.1 Cloud Types

Cloud deployment models are categorized as private Cloud, public Cloud, hybrid Cloud and community Cloud (Casola et al., 2013; Zissis & Lekkas, 2012). Figure 2.2 on the following page illustrates the NIST definition of the Cloud framework (Rong, Nguyen & Jaatun, 2013).

The following are the descriptions of different Cloud types (Rong et al., 2013; Zissis & Lekkas, 2012) :

Private Cloud: The Cloud infrastructure that is operated and managed by private

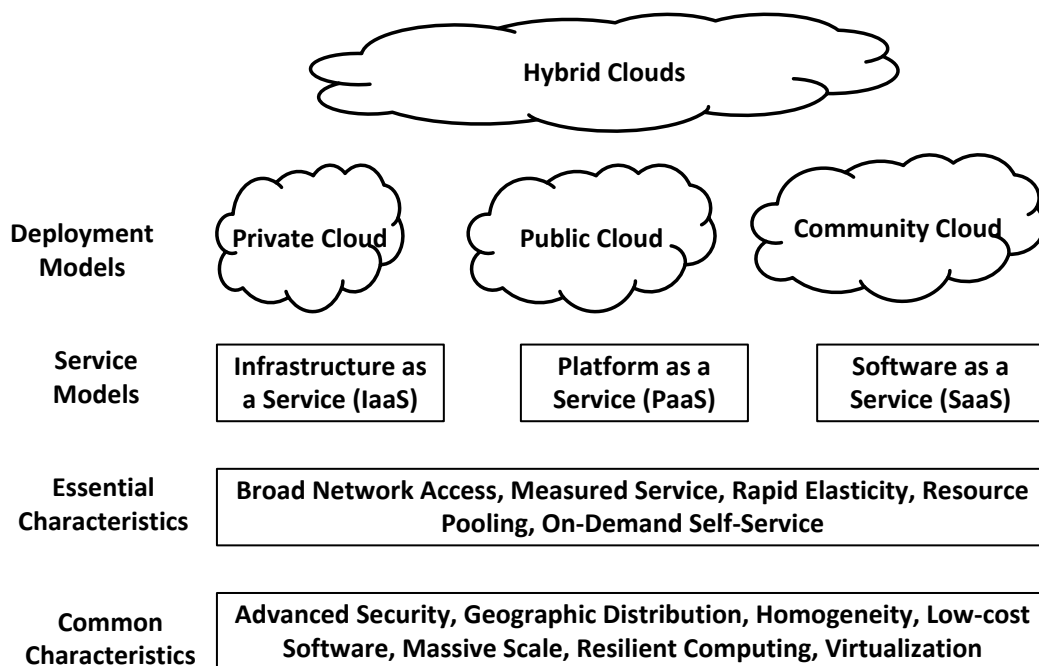


Figure 2.2: NIST definition of the Cloud framework (Rong et al., 2013)

organisations or by an outsourced specialist third party. It can be either on premise or off-premise.

Public Cloud: The Cloud infrastructure or services op for the public. It is normally operated and managed by a single CSP and services are open for subscription by Cloud users.

Community Cloud: The Cloud that is shared by more than one organisations. This kind of Cloud is intended to be used by a community of organizations or entities having common concerns, interests or goals to be accomplished by the use of the Cloud.

Hybrid Cloud: A mix of any of above Clouds (at least two). The Clouds forming a hybrid retain their own distinct characteristics, yet brings portability by means of load-balancing and options to switch among the Clouds of a hybrid Cloud.

2.2.2 Cloud Service Architecture

The deployment of a Cloud architecture is comprised of Cloud deployment models and service models. These models enable different levels of abstraction to enable multiple Cloud users. These service models are the strengths of CC architectures (Fernando et al., 2013). Different deployment models for the Cloud also shape the way security can be perceived in CC. Cloud service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Arshad et al., 2013; Casola et al., 2013; Celesti, Fazio, Villari & Puliafito, 2012; Dukaric & Juric, 2013; Zissis & Lekkas, 2012). The following description explains these service models (Rong et al., 2013; Zissis & Lekkas, 2012):

IaaS: The concept of IaaS entails the provision for a Cloud user to rent infrastructure from the CSP. In IaaS, the infrastructure is deemed to be a service to the end users and hence such name. Users are provided with Cloud resources to enable access to virtual server(s). As the infrastructure is provisioned in IaaS, the user can use the network, processing, storage, computers or servers with the facility of deploying any software set-up. However, in IaaS, only the infrastructure is provided; the platform or OS and the software that would reside on the platform remains under end-user ownership, implying the end-users are required to obtain and maintain them and any licensing issues.

PaaS: The platform or the OS is provisioned to the client in PaaS model. In PaaS-associated Cloud services, the client does not have control over the platform but has total control of whatever can be achieved using that platform. A PaaS model allows a user to configure, deploy, install or un-install any software or service on the platform. In this approach, end-users do not need to worry about the infrastructure and platform licensing issues, but they need to obtain licenses and maintain software that would be installed on the platform. In PaaS, users can

exploit the benefits of the platform but do not manage the platform and cannot access the Cloud infrastructure, network, storage and so on. PaaS has limited scope compared with IaaS.

SaaS: As the name implies, the user is able to use only the software provided by the Cloud provider and cannot extend control beyond the specific software the Cloud user is given access to. The access and management in SaaS are confined to the respective application or software for each Cloud user. Thus, in SaaS, the users are not authorised to manage resources at the IaaS or PaaS level. SaaS has limited scope compared to PaaS.

The above service-related terminologies refer to the provision of technical services and are conceptual. Some authors define a number of other terminologies as Cloud services, for example Data storage as a Service (DaaS), Communication as a Service (CaaS), Security as a Service (SecaaS), Hardware as a Service (HaaS) and Business as a Service (BaaS) (A. N. Khan, Kiah, Khan & Madani, 2013). Database as a Service (DaaS) (Han, Susilo & Mu, 2013) and Application as a Service (AaaS) (Chadwick & Fatema, 2012) are also identified as Cloud service terminologies.

2.2.3 Cloud and Distributed Computing

Che, Duan, Zhang and Fan (2011) describe CC as an offspring of parallel, grid and distributed computing. The convergence of grid computing results in CC (Zissis & Lekkas, 2012). CC provides the platform for multiple domains to co-exist (Han et al., 2013). The Cloud itself is a large distributed network within which the specific infrastructure of any service provider may also be distributed. Subsequently, the Cloud environment embeds the concepts and implementation of the distributed application. The distributed data in a Cloud environment needs to be replicated to ensure availability (Ryan & Falvey, 2012). CC enables large distributed systems as well as highly scalable

and on-demand services (Mackay et al., 2012). CC borrows a number of grid computing characteristics, though a major exception is that Cloud users have less control over location of data (Lombardi & Pietro, 2011), as grid computing does not necessarily mean resources reside in remote location like they do in CC. Scalability, reliability and availability need to be ensured in a distributed environment (Xu, 2012). The distributed nature of CC provides flexibility, while at the same time it increases the risk of security challenges (P. Wilson, 2011).

2.2.4 Cloud and Virtualisation

Virtualisation refers to the technology that enables resource sharing among different parties to reduce overall equipment and management costs (Liang & Yu, 2015). Virtualisation is often considered as the core technology to achieve cost efficiency in deploying Cloud infrastructure. Khan et al. (2013) argue that virtualisation is of benefit to CSPs as it helps in significant cost minimization. Virtualisation helps to achieve cost-effectiveness (Rabai, Jouini, Aiss & Mili, 2013).

CC may be defined without the inclusion of virtualisation, but virtualisation is widely used in Cloud services deployment. This is because virtualisation is the major technological means that brings scalability and flexibility to the Cloud. The back-end of Cloud architecture is comprised of either physical or virtual servers (Rabai et al., 2013).

The multi-tenancy model is a core characteristic of virtualisation technology where one resource can be shared by more than one client. In other words, one physical computer can provide a platform for multiple clients. In multi-tenancy, a single instance of software can be used by multiple tenants (or customers) at the same time (Pal, Mandal & Sarkar, 2015). The multi-tenancy feature helps to deploy several Virtual Machines (VMs) in one physical server – it is important to note that multiple tenants in a multi-tenancy environment may have differing characteristics and perspectives (Azeemi,

Lewis & Tryfonasc, 2013). However, due to its multi-tenancy model with shared technology and infrastructure, the virtualisation platform, the hypervisor, introduces vulnerabilities. The issue with the hypervisors are also to be taken into account in this context. Whether a hypervisor is hosted or native, a security loophole (e.g. hypervisor scape) in the hypervisor places a VM under threat of attacks (A. Patel et al., 2013).

In summary, the concept of CC represents a variation of distributed computing. As the Internet or public infrastructure is a requirement to access remote resources and as CC is distributed in nature, the source of security threats is brought into question. The above discussion suggests that security threats in a Cloud architecture may emerge from numerous focal points. So understanding the scope, impact and importance of security for a Cloud architecture is required.

2.3 Security and CC

In this section, the importance and significance of security for a Cloud architecture is discussed. The discussion addresses the extent to which the Cloud can be affected by threats or vulnerabilities. The degree to which a Cloud architecture requires security and monitoring is also discussed.

2.3.1 Degree to which Threats and Vulnerabilities affect Cloud

A threat can be defined as any event, either intentional or accidental, that yields undesirable consequences to a person, organisation or resources (Newman, 2006). A vulnerability in a system is any existing weakness of the respective system (Onwubiko & Lenaghan, 2007).

Security is one of the key challenges for Cloud architecture (Gonzalez et al., 2012) and it is necessary to review existing challenges (Grobauer, Walloschek & Stocker, 2011). CC facilitates the rapid deployment of remote resources but elasticity and rapid

deployment can be interrupted due to security issues (A. N. Khan et al., 2013). Arshad, Townsend and Xu (2013) say it is important to determine the Level of Severity (LoS) of any Cloud-specific vulnerabilities because Cloud security threats may be the launchpad for an architecture-wide attack.

Virtualisation introduces security challenges that may affect the whole Cloud architecture (Arshad et al., 2013; Han et al., 2013; Liu, 2012). Security issues and loss of integrity are barriers to Cloud deployment (Dukaric & Juric, 2013; Fernando et al., 2013; Chadwick & Fatema, 2012). Cloud-specific security vulnerabilities are discussed in Section 2.4.2 on page 40.

Some of the deployment models in Cloud architecture let customers build customised environments. Cloud users may be able to choose their own OS and software. However, such freedom to choose may lead to security breach in Cloud architecture (Azeemi et al., 2013). In the Cloud, functionality of one level of the infrastructure (e.g. SaaS) and security depends on the robustness of its lower level (e.g. PaaS). This level dependency of Cloud service models makes Cloud security a sensitive aspect to consider. On the other hand, a lower layer may have little control over the security aspects of a higher layer. An example is the IaaS layer which has little control over security functionality beyond its own layer, which may lead to security breaches (Che et al., 2011).

The approach of using outsourced resources leaves CC to be vulnerable (Han et al., 2013) as the resources are under third-party ownership and management. In a Cloud architecture, data can essentially exist outside the scope and beyond the control of an organisation's firewall, and security threats may arise in areas such as lack of control over data, identity audit trails, application security, data recovery, management, privacy, legal issues, business continuity and compliance complexity (Azeemi et al., 2013). Perceptions of security are linked to trust, risks and threats (Khorshed et al., 2012), which implies that security issues are sensitive.

2.3.2 Importance of Security in Cloud

Security is perhaps the most sensitive part for a Cloud architecture because the integrity of a Cloud architecture depends on its security. The scope and impact of security has led to the introduction of security-based Cloud services like Security as a Service (SecaaS) (A. N. Khan et al., 2013).

Zahran (2014) defines holistic security as all aspects of a business requiring security measures functionality. From this, a holistic security vulnerability may pose an architecture wide threat to overall functionality. The significant use of virtualisation and its subsequent holistic security vulnerabilities in Cloud architecture implies that a dedicated and resilient effort for Cloud architecture should be put in place (Arshad et al., 2013). Multi-tenancy and rapid elasticity provide customers with flexibility but introduce security concerns (Azeemi et al., 2013).

Data security and privacy in the Cloud are some of the most important factors to consider (Celesti et al., 2012) and these are still key challenges in CC. Chadwick and Fatema (2012) stress that data security is the biggest challenge for CC. The integrity of services offered by a Cloud provider largely depend on the security aspects of the respective Cloud architecture (Dukaric & Juric, 2013). Efficient management of security is perceived by Cloud users as a key requirement of any CSP (A. N. Khan et al., 2013).

The above discussions point to the fact that CC can achieve its goal of offering flexibility, functionality and elasticity but not without the price of security. Thus, further scope for enhancing Cloud security and subsequently its reliability exists (Esayas, 2012). The consideration of security and threat countermeasures remains a top factor for CC.

The discussion so far indicates that the Cloud is associated with security threats and vulnerabilities. In Section 2.4 on the following page, the threats in other computing approaches are discussed and a mapping between threats in CC and other computing approaches is explained. The discussion suggests that the eventual requirement is for

CC to have security mechanisms that provide a holistic security architecture for safer computing practice. For that, it is important to know the major elements and points in a Cloud architecture that may be prone to security loopholes and attacks.

2.4 Security Points in Cloud Architecture

This section presents a discussion on known threats and vulnerabilities in computing and Cloud-specific threats and vulnerabilities. Security threats in computing and computer communications are experiencing a growth with the popularity and widespread use of the Internet and the World Wide Web (Bottino & Hughes, 2006). Cloud-specific threats arise from the deployment of Cloud-based computing (Grobauer et al., 2011). In this section, the known security threats and vulnerabilities in computing are discussed. Upon outlining the known threats in computing, the Cloud-specific threats and security vulnerabilities are identified. Then the section outlines whether CC threats are similar to or different from known computing threats and vulnerabilities, as well as how these threats and vulnerabilities either differ from or overlap with Cloud-specific ones.

2.4.1 Known Threats and Vulnerabilities in Computing

Computer systems and computer networks have a long history of active and passive threats and security vulnerabilities. Active attacks are where program semantics are observed and modified by attackers; passive attacks are only capable of observing program input and output data (Balliu & Mastroeni, 2009). The active attacks involve modification or destruction of data whereas passive attacks involve data interception to gain confidential information for subsequent abuse or attack (Newman, 2006).

Known computer security threats can be broadly categorised into four major types namely masquerade, interception, modification and DoS (Bottino & Hughes, 2006). A classification of common threats by the Computer Emergency Response Team (CERT)

implies that common threats fall within the categories of Accidental, Intentional, Passive and Active (Bhagyavati & Hicks, 2003). Accidental threats are unintentional, for example, power failure, hardware failure or software failure whereas intentional threats are done to cause damage or loss to the computing environment and to the business or organisation. Active threats, unlike passive threats change the state of a system. However, it does not imply that passive threats are less severe because passive threats are used to sniff and obtain confidential information or credentials on a computer system (Bhagyavati & Hicks, 2003). Gollamann (2010) argues that major security vulnerabilities are associated with OS security, database security, Internet security and software security. Newman (2006) adds intellectual property threats to the list of the threat categories discussed above.

In a networked computing scenario, the servers may be targeted via direct attacks and client-side threats. Web browsers play a key role in client-side threats and vulnerabilities in web based applications, browser plug-ins and content handlers (Hein, Morozov & Saiedian, 2012). Thus, browser security plays a part in CC security (G. Kulkarni, Gambhir, Patil & Dongare, 2012; Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013).

Threats may come in the form of malware, spyware, adware, Trojan horse or virus (Bottino & Hughes, 2006; Arlitsch & Edelman, 2014; Gollmann, 2010). Computer worms are capable of exploiting the vulnerabilities of both servers and workstations (Bowles, 2012).

Newman (2006) describes an integrity threat to be a security issue that, when it occurs, changes the state of data in a computer system from its original state, resulting in integrity loss of the data and/or computer system/network. Newman also describes another threat by naming it as a 'disclosure threat' which is the result of exposing personal or confidential information to unwanted third parties. A disclosure threat is normally a consequence of an integrity threat.

Data flooding can be initiated in a computing environment by uploading unreasonably large files that are subsequently associated with threats like DoS and performance degradation (Bottino & Hughes, 2006).

Identity theft employs various attack techniques to gain access to personal information. A social engineering approach or phishing may help attackers run a Remote Access Tool (RAT) or key logger applications on the end users device or computer to collect information. Botnets are used to send spam or other nefarious acts including DoS. In the Internet age, web-based applications are continually being introduced. The web-based approach creates an environment for attackers and gives a number of ways for attackers to compromise servers. Cross-site scripting and SQL injection are two such examples that help an attacker to exploit server vulnerabilities (Arlitsch & Edelman, 2014). A recent and emerging approach towards threats is termed as ransomware. A ransomware is defined by TrendMicro (2017) as “a type of malware that prevents or limits users from accessing their system, either by locking the system’s screen or by locking the users’ files unless a ransom is paid”. An example of such ransomware is Petya (TheGuardian, 2017) In Section 2.4.2, known computing threats that may equally be treated as Cloud specific threats are discussed.

2.4.2 Cloud-Specific Threats and Vulnerabilities

As defined by Newman (2006), the major parts and approaches within a computing and computer network environment that are prone to security vulnerabilities are client PCs, remote access users, Internet access and servers. It may be argued that CC is a conceptual approach utilising existing computing technologies. Therefore the argument is made that known computing threats and vulnerabilities can potentially be a Cloud-specific threat but not essentially vice versa.

How to categorise threats is under debate. Grobauer, Walloschek and Stocker (2011)

argue that threats should be categorized as Cloud-specific and non-Cloud specific; whereas Liu (2012) argues that all traditional threats and vulnerabilities are equally applicable to CC.

For example, some authors argue that non-Cloud-specific threats and vulnerabilities are listed as Cloud-specific threats or vulnerabilities (Grobauer et al., 2011). As CC is an amalgamation of other computing approaches, then those threats and vulnerabilities apply to the Cloud (Vaquero, Roderio-Merino & Moron, 2011). Fernandes et al. (2014) state that a number of security issue (e.g. packet sniffing, social engineering, malware, Port scanning, IP spoofing) on the Internet are inherited by CC. Chirag et al. (2013) suggest that vulnerabilities related to Internet protocols (e.g. RIP attack, flooding, ARP spoofing, DNS poisoning, IP spoofing, man-in-the-middle-attack) may be used as means to attack Cloud systems. A number of security concerns that are listed as Cloud-specific concerns are not new threats but instead they overlap Cloud and non-Cloud specific threats. For example, data privacy and security, external threats (e.g. man-in-the-middle attack, IP spoofing, packet sniffing), Distributed DoS (DDoS) are some examples of supposed Cloud threats (Yu, Powell, Stembridge & Yuan, 2012) that are common to non-Cloud-based computing environments. Kulkarni et al. (2012) also denote DDoS and poorly configured Secured Socket Layer (SSL) threats to be Cloud security concerns but are common in other computing approaches. Srinivasan et al. (2012) list cryptography and key management as a Cloud threat with an additional note that these are not Cloud-specific vulnerabilities but rather traditional computing threats that also affect the Cloud and thus are security concerns for CC as well.

CC is a very broad term that includes recent developments in Internet-based computing (Roberts & Al-Hamdani, 2011). All the security concerns related to the Internet are subsequently factors for CC security as well (Liu, 2012). Due to having the Internet as the general backbone for Cloud provision, TCP/IP vulnerabilities may emerge as potential threats (Modi, Patel, Borisaniya, Patel & Rajarajan, 2013). Apart from these,

web services play a key role in accessing Cloud resources and introducing security concerns into the CC context (Fernandes et al., 2014). All these characteristics of CC make it difficult to distinguish between Cloud-specific and non-Cloud specific threats. Chirag et al. (2013) draw special attention to HTTP protocol vulnerabilities in accessing Cloud services. A new concept named Just Enough Operating System (JeOS) provides access to Cloud facilities through web browsers (Roberts & Al-Hamdani, 2011). JeOS is accessed via web-based services – this points out that web services and their associated threats or concerns are crucial and sensitive in a Cloud security context.

When CC security is taken into account, the threats and vulnerabilities are more perceptual than technical. For example, data leakage is described as one of the Cloud-specific vulnerabilities (Sabahi, 2011) which can be exploited by any technical attacks normally known as non-Cloud specific threats. Cloud Security Alliance (CSA) (2013) describe nine Cloud threats: data breaches, data loss, account or service hijacking, insecure interface or APIs, denial of service, malicious insiders, abuse of Cloud services, insufficient due diligence, and shared technology vulnerabilities. None of these threats are new (Roberts & Al-Hamdani, 2011; Liu, 2012; G. Kulkarni et al., 2012; Fernandes et al., 2014). Fernandes et al. (2014) propose a Cloud security taxonomy and list eight major categories: trust, compliance, legality, network, access, virtualisation, Internet and services, storage and computing, and software. The taxonomy includes threats similar to those listed by CSA.

CC brings novel security concerns that demand distinct attention and countermeasures (Roberts & Al-Hamdani, 2011). Some of the identified novel Cloud threats are XML signature attacks in the form of wrapper attacks, browser security, flooding, reputation fate sharing, side channel attacks, loose control over data and Internet dependency. In XML signature attacks, a duplicate fragment of XML code is injected with additional malicious code to force a computer to perform unintended tasks (Roberts & Al-Hamdani, 2011; Khalil, Khreishah, Bouktif & Ahmad, 2013; Modi, Patel, Borisaniya, Patel, Patel

& Rajarajan, 2013). Srinivasan et al. (2012) add Cloud and CSP migration to the list of security threats.

It is rare to find an article on Cloud security that excludes virtualisation from the list of Cloud threats and vulnerabilities. Virtualisation is a key technology in CC and at the same time, it is one of the most crucial security concerns for Cloud architectures (Roberts & Al-Hamdani, 2011; Srinivasan et al., 2012; Alfath, Baina & Baina, 2013; Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013). Virtualisation enables a Cloud server to create and destroy virtual servers on-the-fly which in turn introduce security concerns for identity management (Bouayad, Blilat, el houda Mejhed & Ghazi, 2012; G. Kulkarni et al., 2012; Behl & Behl, 2012; Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013). The concern gets more challenging when IP addresses are dynamically assigned to newly-created virtual servers within a Cloud architecture (Srinivasan et al., 2012).

Virtualisation enables hardware sharing which leads to a vulnerability known as reputation fate sharing. In such cases, the reputation of user A can be affected due to user B sharing the same hardware with user A (Roberts & Al-Hamdani, 2011; Srinivasan et al., 2012). Due to virtualisation, the logical segregation of data is also an area of security concern (Srinivasan et al., 2012) as communication among different VMs may open up security loopholes to accelerate threats like reputation fate sharing.

A direct vulnerability due to the use of virtualisation and subsequent hardware sharing may emerge from inefficient data sanitisation. Sanitisation means the process of wiping out the data from a resource before provisioning the respective resource to another party. If done inefficiently, sanitisation may lead to the threat of data loss or data disclosure. The pooling and elasticity characteristics of CC helps to reallocate resources to different parties quickly, which may be associated with inefficient sanitization and thus act as a threat for Cloud architectures (Fernandes et al., 2014).

The use of virtualisation and its multi-tenancy model bring security concerns to CC

(Bouayad et al., 2012; Behl & Behl, 2012; Alfath et al., 2013; Khalil et al., 2013; Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013; Vaquero et al., 2011; Fernandes et al., 2014). Examples where virtualisation related threats and vulnerabilities may exist are untrusted components, transparency, VM cloning and software duplication (Pearce, Zeadally & Hunt, 2013). Chirag et al. (2013) describe virtualisation and its multi-tenancy to be a direct threat to Cloud privacy and security. The use of virtualisation and the subsequent hardware and other resource sharing also leads to the risk of backdoor channel attacks (Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013) or side channel attacks (Vaquero et al., 2011). A side channel attack is when an attacker uses one VM to compromise another to intercept data to and from the compromised VM. Remote storing of data in someone else's server results in loss of control over data in a Cloud environment (Behl & Behl, 2012; Alfath et al., 2013), leading to privacy and security concerns (Gritzalis et al., 2014b). Side channel attack is described as the most significant Cloud threat, and is coupled with trust factors for CC. Beckers, Cote and Goeke (2014) assert security and privacy as an essential factor for CC because customers of the Cloud have no choice but to entrust the CSPs as soon as they decide to move to the Cloud. Shaikh and Haider (2011) state that lack of proper security may emerge as a major Cloud disadvantage while trust is one of the most crucial ones. Liu (2012), Chirag et al. (2013), Vaquero et al. (2011), Fernandes et al. (2014) and Sumter (2010) also agree that security and privacy are the biggest concerns in CC.

2.4.3 Cloud Threat Taxonomy

As part of the presented research, a taxonomy for Cloud threats is developed (Ahmed & Litchfield, 2016) based on existing literature and recent Cloud breaches. To provide a sound epistemological foundation for the study, existing research is used to develop the taxonomy. For example, in Gonzalez et al. (2012), threat-specific viewpoints

presented are architectural, compliance, and privacy issues. In Hashemi and Ardakani (2012), a security taxonomy for CC presents four categories as follows: infrastructure, application, platform, and administration. Other research categorises CC threats as responsibility ambiguity, protection inconsistency, evolutional risks, supplier lock-in, business discontinuity, license risks, bylaw conflicts, bad integration, hypervisor isolation failure, service unavailability, data unreliability, abuse of rights of the CSP, shared environment, and use of insecure Application Program Interfaces (APIs) (G. Kulkarni et al., 2012).

It may be noted that, as well as coming from traditional network threats, CC threats emerge from its deployment model (Chou, 2013). This factor makes the deployment model one that needs consideration. Additionally, threats arising from management issues in CC have been identified, for example, costs, benchmarking, change management, legislation, Service Level Agreement (SLA), lack of interoperability, information retrieval, information localisation, standardisation, single point of failure, service availability, and security issues (Cardoso & Simões, 2012). In Soares et al. (2014), eight threat categories for CC are listed: security issues identified by organizations; deployment and service delivery model security; software-related security issues; data storage and computational security issues; virtualisation security issues; networking, web, and hardware resources security issues; access; and trust security issues. On the other hand, the discussion in (Singh & Shrivastava, 2012) identifies a threat taxonomy on the basis of the classes of participants within a Cloud infrastructure and the specified participants are: service users, service instances, and Cloud providers. As a further categorisation, six root categories are outlined: service-to-user, user-to-service, Cloud-to-service, service-to-Cloud, Cloud-to-user, and user-to-Cloud. Considering technical aspects, five categories of concern to Cloud security are identified: hardware components, VM manager, guest OS, applications network and governance.

From the above, it may be noted that the identification of threats in a Cloud environment are subject to the context in which they are viewed and it is from this position that the taxonomy presented here is taken. Concerns over the current level of understanding about CC security has moved some to make comment on the state of CC security as “confusing” (Fernandes et al., 2014). Thus, this taxonomy is more generally applicable and therefore generalisable. To facilitate unambiguous study and categorization of Cloud threats, the taxonomy includes all the genres of threats to CC. Consequently, new and emerging threats may be identified, categorised, and subsequently countermeasures for the respective threats may be provided.

In the following discussion, a context for security concerns in CC is created, and Cloud deployment models are presented; the next section addresses virtualisation and its significance and scope within Cloud infrastructure deployment. Nine case studies of Cloud breaches that demonstrate some of the threats identified are discussed in the taxonomy. Following that, security concerns specific to CC are presented and categorised into a generalised Cloud threat taxonomy. Then, recent cases of Cloud breaches are applied to the taxonomy to determine whether the taxonomy holds true.

Case Studies

In order to build a taxonomy for further analysis of threats against Cloud-based computing environments, nine case studies of Cloud breaches have been analysed. The method of analysis is a qualitative assessment of the type of breach of security and the extent to which the breach has affected stakeholders. The breaches discussed here have been expanded with more recent exploits to demonstrate the validity of the taxonomy. The purpose of the taxonomy is to ascertain what types of exploits have been successful (as opposed to vulnerabilities that may not have been exploited) and to provide a basis for requirements gathering in the design of a Cloud-based security system. Along with a mapping of attacks into the taxonomy, the following discussion presents the case studies

that are summarised in Table 2.1. To summarize the threats, the relevant literature is presented in Table 2.2 on the next page, which includes a mapping of the taxonomy to threats identified.

Table 2.1: Summary of Attacks in terms of the Taxonomy

Description of attack	Threat Type in Taxonomy	Case study
Brute-force password cracking, claimed to be targeted attack instead of an architecture-wide to obtain specific users' password and credentials.	Social Context, Competence	Apple iCloud breach
Hacker's signed up for VM and then accessed co-located SONY's VM on Amazon EC2 server. A cross VM side-channel attack.	Computing Services, virtualisation	SONY server attack through Amazon VM
Weak code execution vulnerability helped attackers to access to access a total of 76 million users' and 7 million small businesses' personal information (names, email addresses, and physical addresses).	Software Tools, Web Services, Competence	JPMorgan server hack
Unprotected for 4 hours, any Dropbox account could be accessed by using any password during the period.	Lack of Competence, Trust	Dropbox
Snapchat server hacked and claimed to be done so by reverse engineering the API by a third party app, which also resulted in non-compliance of end user license agreement for affected Snapchat users.	Social Context, Competence, SLA misinterpretation	Snapchat
Attackers took the vendor's services down by installing malware on end users' devices. Vendor claimed the loophole was in end users' computers, and had nothing to do with their Cloud architecture.	Social Context, Competence, Software tool, web service, Cloud platform, computing services, virtualisation	Spark

continued ...

Summary of Attacks

... continued

Description of attack	Threat Type in Taxonomy	Case study
Login details of the users were stolen and sold on Dark Net. Uber points the responsibility was on the users, who used same login credentials on different sites.	Social Context, Competence, Trust, Regulation	Uber taxi
Unusual activities detected on server that resulted some user data loss. The network-wide unexpected breach required the servers to take to “lock-down” mode.	Cloud Platform, Social Context, Competence	LastPass
24 million users’ data exposed yet no specific information on how the attack emerged.	Competence	Zappos

Table 2.2: Threats in Literature Categorized according to the Taxonomy

Category in Taxonomy	Threat	Reference
Trust	Trust, trust management	Gonzalez et al. (2012); Guo, Sun, Chang, Sun and Wang (2011); K. Khan and Malluhi (2010); Noor, Sheng, Zeadally and Yu (2013); Roberts and Al-Hamdani (2011)
Compliance, regulations	Standards, Compliance measures	Srinivasan et al. (2012)
SLA misinterpretation	Proper SLA provisioning, Poor understanding of SLA	Casalicchio and Silvestri (2013); Emeakaro et al. (2012); Srinivasan et al. (2012); Sun, Singh and Hussain (2012)
Social context, competence, specialization	Social Engineering	Krombholz, Hobel, Huber and Weippl (2013); Robling and Muller (2009); Thornburgh (2004)
Computing services	Outsourcing Cloud services	Duncan, Creese and Goldsmith (2012)
Internal infrastructure	Insider attack	Duncan et al. (2012)
External networks	Wireless link exploitation	Vikas, Gurudatt, Pawan and Shyam (2014)
Local platform	Poor user control on data in smartphones	Fernando et al. (2013)

continued ...

Threats in Literature Categorized

... continued

Category in Taxonomy	Threat	Reference
Cloud platform	Insider attack	Duncan et al. (2012)
Network protocols	Peer-to-peer protocol vulnerabilities	Tong, Xiong, Zhao and Guo (2013)
Virtualisation	Hyper-jacking, cross-VM side channel attack	Grobauer et al. (2011); Perez-Botero, Szefer and Lee (2013)
Software tools	Application vulnerability	Grobauer et al. (2011)
Web services	Web based services, HTTP vulnerabilities	Grobauer et al. (2011); Modi, Patel, Borisaniya, Patel, Patel and Rajarajan (2013)
Security mechanisms	Traditional weakness in cryptography, Theft of cryptographic key, Authentication issues	Fan, S and Huang (2013); Fernando et al. (2013); Grobauer et al. (2011)
Mobile computing	Vulnerable applications spread through mobile devices	Vikas et al. (2014)

The first case is Apple's iCloud service that provides a consumer-level service to back up smart-phones' content (such as music, photos, and data) (TechTimes, 2014). Hackers breached the Cloud service on September 1, 2014 and the personal photos of celebrities were made publicly available (PCWorld, 2014). The hackers used a brute force password cracking approach to gain unauthorised access to the celebrity photos. Apple refused to take responsibility, stating that it was a targeted attack and not a Cloud breach (TechTimes, 2014; Journal, 2014). Apple claims the incident was not an architecture-wide breach of iCloud but a focused attack on specific user accounts where unauthorised access was gained by obtaining passwords, security questions, and usernames.

The iCloud case represents a combination of factors (human and technological) that include the ease with which passwords may be discovered by brute force methods and the simplicity of passwords selected by users. In the examples presented above, one

cannot simply point the finger of blame at the user because passwords are themselves the most basic form of security. A process that is dependent upon knowing answers to security questions is inherently weak. In a similar case in early 2015, the login details of Uber taxi users were stolen and sold on the Dark Net (TheGuardian, 2015). This allowed buyers to obtain user credentials and to access Uber services. Uber denies that any attack on its Cloud servers occurred, and lays the blame on users, saying that they should not use the same login details on multiple sites.

In Contrast, online password manager, LastPass detected unusual activity in its Cloud servers and some user data was stolen (Paul, 2015). The servers were put into "lockdown" mode and the network-wide malicious activity is not considered a bad breach because the effects of the attack were limited by the option to use multi-factor authentication or not using password reminders.

In April 2011, Sony Corporation's online entertainment system experienced an attack that netted the attackers the second largest haul to that date. The personal accounts of more than 100 million Sony customers were exposed. In the attack on Amazon's Cloud-based web servers and by using an alias, hackers signed up to Amazon's EC2 service and from there they were able to successfully access co-located VMs on the same cluster (Bloomberg, 2011a, 2011b). The attack was very carefully planned and executed. The opportunity to use the Cloud servers helped the attackers to maintain anonymity. This case differs from the previous insofar that the method of attack is far more sophisticated and requires significant expertise and knowledge of the Cloud infrastructure. The method of approach resembles a cross VM side channel attack which, to be successful, needs multiple stages to be executed in order for the attacker to get access to the VM encryption keys. The previous case, by using brute force techniques, could be achieved by any attacker who has the patience to wait for a password to be successfully found.

The next case reports unauthorised access to 80 million account holders of JPMorgan by means of a weak code execution vulnerability (CNN, 2014). A total of 76 million users' and seven million small business' personal information (names, email addresses, and physical addresses) was accessed by the attackers (Computerworld, 2014). JPMorgan claim no unauthorised access could be made to account-related information, user IDs, and date of birth or account numbers, and no fraudulent customer activities were found as a result of the incident. It was users who accessed the service through websites or smart-phones that were affected. It is probable that the only abuse of the leaked information would be by spammers. For this case, the issue appears to be focused around vulnerabilities in the computer–user interface.

Case four occurred in mid-2011, when Dropbox experienced an attack. The website of the Cloud provider was left unprotected and exposed, leaving servers containing the personal sensitive data and information of a massive number of users exposed. An estimated 25 million users use Dropbox to store their personal contents such as images, documents, videos or other types of files. During the breach, Dropbox was unprotected for four hours, giving anyone the opportunity to access any account with any password. To give its users access to the system, the provider had traded security for ease of use by requiring only a simple password instead of having complex encryption keys; and keeping the encryption and decryption process within its own servers (CNN, 2011). Similar to Case one, but more extreme, this case represents a failure of security policy and management. Needless to say, this is no longer the case today.

The fifth case involves Snapchat, an instant photo-sharing application. It is claimed that Snapchat's Cloud servers were hacked in October 2014. The claim is that either the service provider's Cloud servers or its end-user application, or both, were hacked. Snapchat denies the claims and points to a third-party vendor that had applied a reversed engineering approach to Snapchat's API to store users' data from Snapchat's servers.

This third-party app, when used, made the Snapchat users non-compliant with the end-user license agreement (TheRegister.co.uk, 2014). Such a case represents a complex situation that is likely to become more frequent as more use is made of API's that may offer up vulnerabilities for exploitation. The SnapChat case provides an example of how technological factors may be combined with human factors to produce conditions ripe for an attacker to use.

Another case involves Spark, a Telecom service provider in New Zealand. In September 2014, as the consequence of a cyber attack, the vendor's Cloud services were significantly interrupted and subsequently shut down and broadband services became unavailable. The attackers, believed to be international cyber criminals, had installed malware on Spark's end users' computers. The malware was equipped with malicious code to effectively launch a denial of service attack. The attack subsequently generated heavy traffic to create a situation of service unavailability. Spark's claim is that its own Cloud architecture did not have any fault, but rather it was the end users' computers that were affected and were used to launch the attack (Stuff.co.nz, 2014). This case represents what is perhaps one of the greatest weaknesses in a Cloud architecture, and that is the human factor. In all of these cases, the underlying theme appears to be that this is the factor that either creates the conditions for or allows vulnerabilities to be exploited.

Shoe retailer Zappos' data breach in January 2015 exposed 24 million users' data to attackers (Schwartz, 2015). There is almost no information on how the attack emerged (Bradley, 2015). However, the case illustrates the importance of focusing on the exploitation of zero day vulnerabilities. The service provider or the computing services Zappos uses to deliver its service appear to be at fault for not addressing vulnerabilities.

The above case studies were recent at the time the threat taxonomy was developed. Cloud breaches continue to emerge where some of them are associated with massive negative impact. The more recent and emerging Cloud breaches can be mapped to the

developed taxonomy - thought the taxonomy was developed before the following more recent breaches took place, comparing them with the above case studies would give a clear indication on how these new case studies fit within the taxonomy. Few more recent cases of Cloud breach are discussed below.

“On December 30, 2016, ESEA, one of the largest video gaming communities, issued a warning to players after discovering a breach. At the time, it wasn’t known what was stolen and how many people were affected. However, in January, LeakedSource revealed that 1,503,707 ESEA records had been added to its database and that leaked records included a great deal of private information: registration date, city, state, last login, username, first and last name, bcrypt hash, email address, date of birth, zip code, phone number, website URL, Steam ID, Xbox ID, and PSN ID" (IdentityForce, 2017). This is also reported by Ragan (2017).

“IHG, the company that owns popular hotel chains like Crowne Plaza, Holiday Inn, Candlewood Suites, and Kimpton Hotels, announced a data breach that affected 12 of its properties. Malware was found on servers which processed payments made at on-site restaurants and bars; travellers that used cards at the front desk did not have information taken. The malware was active from August 2016 to December 2016 and stolen data includes cardholder names, card numbers, expiration dates, and internal verification codes. Some targeted locations include Sevens Bar & Grill at Crowne Plaza San Jose-Silicon Valley, the Bristol Bar & Grille at the Holiday Inn in San Francisco’s Fisherman’s Wharf, InterContinental San Francisco, Aruba’s Holiday Inn Resort, and InterContinental Los Angeles Century City" (IdentityForce, 2017). This is also reported by Osborne (2017).

“Dun & Bradstreet, a huge business services company, found its marketing database with over 33 million corporate contacts shared across the web in March 2017. The firm claims its systems were not breached, but that it has sold the 52GB database to thousands of companies across the country; it’s unclear which of those businesses

suffered the breach that exposed the records. Millions of employees from organizations like the U.S. Department of Defence, the U.S. Postal Service, AT&T, Wal-Mart, and CVS Health had information leaked, and the database may have included full names, work email addresses, phone numbers, and other business-related data" (IdentityForce, 2017). This is also reported by FoxNewsTech (2017).

"OneLogin, a San Francisco-based company that allows users to manage logins to multiple sites and apps through a cloud-based platform, has reported a troubling data breach. OneLogin provides single sign-on and identity management for about 2,000 companies in 44 countries, over 300 app vendors and more than 70 software-as-a-service providers. A threat actor obtained access to a set of Amazon Web Server (AWS) keys and used them to access the AWS API from an intermediate host with another, smaller service provider in the US. The attack began at 2am PST on May 31 and was shut down by 9am. Customer data was compromised during this time, including the ability to decrypt encrypted data. The investigation is ongoing and the full extent of the breach is still unknown" (IdentityForce, 2017). This is also reported by BBC (2017).

CC Threats

It may be argued that CC is a concept with substantiated properties rather than being a technology itself. A Cloud service operates within social and organisational settings, thus considerations to take into account are technical and human soft factors such as socio-technical issues, cultural and social contextual issues, domestic or international regulations, and users' computer literacy.

While security concerns and threats to CC are important (Gonzalez et al., 2012), it may be difficult to determine that any breach is a failure of a Cloud technology. That is, CC consists of a range of contemporary computing and Internet technologies that have been grouped together in order for people to understand the context they operate in. CC specific threats may not be limited to a Cloud architecture. Instead, one must consider

all substantial and conceptual properties of CC such as the Cloud architecture, users, and intermediate provisioning (Internet or any public or private infrastructure).

To define “threat” for CC, the cases have been analysed to produce a genre-based top-down threat taxonomy specific to CC (see Figure 2.3 on the following page). A high-level categorisation of a threat taxonomy provides for most computing technologies that fall within the operational or architectural context of CC. Further, a threat to the security of a CC environment may be seen as a part of a specific Cloud architecture. This implies that such a threat is a technological and, to some extent, a regulatory and location transparency-related problem.

Figure 2.3 on the next page illustrates that at its highest level, CC threats can be categorised as technological factors (or hard threats) and human factors (or soft threats). These are discussed in the following sections. In the computing environment, technological and human factors play an equally important role, as mentioned in several previous researches (Haniff & Baber, 1999; Hawkey et al., 2008; Kueppers & Schillingno, 1999; Mohamadi & Ranjbaran, 2013). This approach separates designed and accidental threats through hardware and software as technology from deliberate and non-deliberate threats by human actions.

Human Factors

The reference to “soft threats” above relates to threats that arise from human-centric actions that threaten a Cloud infrastructure. Such threats might be associated with government regulations for any given region or country, the lack of data security and consistency in a location-independent CC case, social engineering, poor computer literacy of service consumers, level of trust among various Cloud stakeholders and might be a direct influence on cultural or social norms, compliance or lack of well-defined compliance standards. This does not relate to incidents where data security has been breached as a consequence of, say, a staff member who inadvertently leaves a



Figure 2.3: Threat Taxonomy for CC (Ahmed & Litchfield, 2016)

removable drive in a public place.

As dynamic security challenges emerge, two human factors, compliance and competence, are required for Cloud providers and end users. If compliance is not in place, then out-of-standard procedures are allowed, and threats to security may occur. The competence of IT practitioners and management is a factor where users, developers, and Cloud providers demonstrate good practice to prevent unwanted events.

A constant threat to Internet security and therefore the Cloud environment is social engineering (Krombholz et al., 2013). The social context refers to the use of CC within

the community, which in turn is associated with other factors such as trust and social engineering, specific human-related security issues such as a lack of awareness or training, or lack of vigilance or caution when using Cloud services. The social context and related social engineering have been referred to as the dark art that poses a severe threat to confidentiality, integrity, and authenticity of information (Thornburgh, 2004) and alternatively defined as the act of manipulating people to extract or gain access to confidential information (Robling & Muller, 2009). This may have greater effect where a lack of competence may result in situation that would aid in achieving the aims of the attacker.

In the Snapchat case study, image access is limited after a short time although images exist on intermediate servers as they move through networks. To bypass the terms and conditions set by Snapchat and to get and store images from Snapchat servers, users install a third-party application. The application intercepts the data or images and in this situation, images that the originator did not intend to last were captured and, in some cases, were used to embarrass or compromise individuals. Ignorance of how the system handles data lulled users into a false sense of security. This also illustrates how someone with malicious intent is able to step around a socially ordered agreement.

It is known that phishing messages purporting to be from banks and other institutions are actually sent from malicious sources. While people are often warned not to click on attachments, they still do. In cases, the users click on a link to a site masquerading as an official site but that contains an Adobe Flash file containing malicious code or malware. The above practice may be attributable to a lack of competence or social engineering. Whatever the case, the effects on large numbers of users who are similarly duped enables the creation of botnets in which a botnet master is able to control functions on computers en masse. In the case of New-Zealand telecommunications company Spark, a significant outage occurred due to a denial of service attack on at least one of their servers. Spark's customers, not the company itself, may have been the target,

but the result was that thousands of users lost access to broadband Internet services. The service provider blamed users who downloaded and installed malware into their computers by opening infected email attachments.

The taxonomy does not present elements that are mutually exclusive, that is, a specific Cloud breach may incorporate a number of threats. The case of JPMorgan is such an example in that while there has been no evidence of abuse of users' accounts, stolen data may have been sold to criminals or spammers. Thus, trust of an organisation tasked with maintaining the privacy of user data is brought into question, regardless of whether the reason behind such a breach is technological (e.g. web service, Cloud platform, computing services, and virtualisation) or human factors (e.g. social context and competence).

Location transparency is a security concern linked to social or regulatory contexts rather than technological issues. If data from one region (e.g. nation and jurisdiction) are transferred to any other region, there is no guarantee that the data are being treated in the same way as the source. This includes the level of security as well as retention and processing of data. Ideally, regulations that address data management would be consistent across jurisdictions. However, this ideal has yet to be reached.

Due to the complexity of CC architectures, SLA provisioning needs careful assessment (Casalicchio & Silvestri, 2013). Also, misinterpretations of the SLA are related to failures in Cloud computer security (Srinivasan et al., 2012). For example, there is a tendency for customers to make an agreement, perhaps through a lack of understanding of legal jargon, without actually reading the end-user license. Thus, it is important to make a commitment to the SLA to provide an assurance that conditions are met (Sun et al., 2012). Conversely, it may be argued that a lack of commitment leads to a lack of monitoring of an SLA (Emeakaroha et al., 2012).

In any case, whether it is through SLA confusion or expectation, trust in the Cloud is a major issue and trust management and security as a key challenge (Gonzalez et

al., 2012). Trust is a state that helps one party to keep faith and be reliable based on transparent control practices, ownership, and security in a Cloud environment (K. Khan & Malluhi, 2010). While trust derives from the social sciences, in heterogeneous computing a relationship to security is implied (Guo et al., 2011). Users of Cloud-based services are expected to trust the Cloud provider with sensitive, personal and confidential data (Roberts & Al-Hamdani, 2011). A lack of trust is a conscious state and while it may not be a conscious choice, it is one that is held by the service consumer. This is a technology independent viewpoint and while it relies on a service–consumer perception, trust is affected by the level of policy or procedure development at the micro- or organisational level or at the macro- or regional level, the robustness of a regulatory framework. A crucial concern is how standards are applied and whether compliance measures are sufficiently robust in a given region (Srinivasan et al., 2012).

Regulation and compliance conformity differ from procedure where a lack of adequate governance affects the level of trust a customer will hold in a Cloud provider. The Dropbox case provides an example of how, in CC, trust can be affected through either a lack of competence or a failure to predict security requirements, and how security should be integrated into a Cloud service.

Of the cases presented, not all are direct attacks on the Cloud architecture. In the Apple case, user accounts were hacked through brute-force methods and the damage was limited. Though Apple provides strong security mechanisms to safeguard its Cloud architecture, weak user passwords leave accounts and possibly other systems open to attack.

Technological Factors

Technological factors refer to threats other than human or social factors. It may be argued that, in the computing environment, the factors or threats in this category fall into two categories: (1) hardware-related threats that relate to the Cloud infrastructure and

network and, (2) software-related threats that relate to platform and application resources above the Cloud infrastructure. Across both categories, Internet-based vulnerabilities provide security concerns because the Internet is the primary means of accessing Cloud resources (Grobauer et al., 2011).

Additionally, virtualisation, web services, and application and cryptography are associated with vulnerabilities (Grobauer et al., 2011). virtualisation provides a number of software-based security threats (Perez-Botero et al., 2013), for example, denial of service, and hypervisor exploits such as hyper-jacking, cross-VM side channel attacks and hypervisor escape.

Web services introduce challenges to security, for example, HTTP vulnerabilities that present threats while users access Cloud services (Modi, Patel, Borisaniya, Patel, Patel & Rajarajan, 2013). Threats generally include Structured Query Language (SQL) injection, cross-site scripting, lack of web site security, directory traversal, lack of AJAX security from poor programming, Apache web server vulnerabilities, and lack public Cloud provider security measures in tools such as WordPress.

While cryptography is applied when other measures cannot assure security, for example, when data may be encrypted before being sent to a Cloud service, cryptographic mechanisms may be associated with weaknesses (Fan et al., 2013), for example, successful interception of data in man-in-the-middle exploits and subsequent decryption of intercepted data. Another example is, the theft of cryptographic keys during successful cross-VM side channel attacks.

Mobile computing is becoming a common means for distributed and general computing (Chow et al., 2010), so that mobile CC is becoming an inherent part of the total CC practice (Cheng, 2011), and this has led to the concept of Bring Your Own Device (BYOD) (Krombholz et al., 2013). Security vulnerabilities in mobile technologies are now becoming more apparent, especially in the more open Android development space. The provisioning of mobile computing introduces application and network-based threats

(Fernando et al., 2013; P. Kulkarni & Khanai, 2015; Vaquero et al., 2011).

Threats can be introduced to the Cloud environment internally. For example, maintenance services such as incident response or routine maintenance provides opportunities for a person (insider or outsider) to access resources they are not entitled to. Failing or malfunctioning hardware provides the opportunity for potential security breaches. However, organisations may not be willing to reveal a Cloud breach or the actual reason for a breach for fear of losing trust or goodwill from stakeholders. A number of breaches listed may incorporate threats that arise from network or hardware component failures, poor configuration, inconsistent maintenance, or poor management of incident response. Vulnerabilities in peer-to-peer network protocols may therefore be exploited (Tong et al., 2013). Additionally, an insider attack may be carried out through unauthorised on-premise or remote access to Cloud resources as a consequence of outsourcing Cloud services to third parties (Duncan et al., 2012).

As an example of a local or end-user platform-specific issue, smartphone apps often require access to more data that are necessarily required to function, so that users of smart-phones have poor control over data stored on phones (Fernando et al., 2013).

Expectations of social or cultural behaviours are often played out in the cyber-world and are socio-technical in orientation. In the Spark case study, through a complex interplay of social engineering, using end users' computers, a successful attack on the company's server network was launched.

The Uber example represents a case of social context and competence. If users were not aware and shared their login credentials with someone who has then taken advantage of the opportunity to also make use of the user account, then the threat is one of social engineering and therefore has a social context. If the misuse of the user information is due to a technological issue, as a consequence of a third party that is considered a credible and trustworthy service provider and with whom the user has shared Uber login credentials, then the threat can be categorised as trust, regulations,

and competence. Trust is a factor between Uber and any other service provider with whom users shared the login credentials. Also, the third-party service provider may show a lack of competence.

Aligning cases to the taxonomy may involve more than one category. In the case of LastPass, while the breach is technological (e.g., via the Cloud Platform), the users played a part through their social context and level of competence (thus it was related to human factors).

Implications of the Taxonomy

The taxonomy is generalised and genre-based and thus provides understanding of the nature of existing and new threats in the Cloud environment, unlike other taxonomies that tend toward specific problem or application areas. As new threats emerge, the taxonomy may be applied to specific problem areas and its value is realised. For example, the taxonomy has been applied to the modelling and design of a distributed application architecture to provide security functions for a CC environment (Ahmed, Litchfield & Sharma, 2016). This solution is created to provide a fully redundant, distributed security system with no single point of failure and so far as the attacker is concerned, the system appears to be everywhere. In this instance, the taxonomy identifies where threats are likely to emerge and aids in the classification of threat types when designing functional requirements.

Other potential uses of the taxonomy are in social sciences research, so that researchers can properly separate hard and soft factors from threats or attacks. For example, human factors are seen to be a significant factor in threats that include the distribution and use of embedded malware and ransomware. The taxonomy provides the social science researchers with the opportunity to propose solutions appropriate to usage patterns by human actors such as educational programs or recommendations for the design of corporate computer use policies. Also in the enterprise space, the taxonomy

is useful in the design and planning of enterprise security policy. For example, again, the human factor is often seen as the biggest risk to enterprise systems, whether by misuse, ignorance, or omission. However, policy ought to include the management of technological factors. In this instance, the IS researcher is inclined to consider what technological aspects have been omitted from policy and what impact that may have in the event that the omission is exploited.

The area of trust management is another in which the taxonomy may be applied. Trust is a complex issue and the individual factors that influence the presence or absence of trust combine to make the accumulation of trust in a service difficult to predict. Therefore, it is difficult to put in place plans or policies that will confidently lead to a trust-based relationship between the service provider and consumer.

CC incorporates existing and new computing technologies, and threats to services are inherited from existing technologies and new threats are introduced. The determination of threats is made difficult due to the range of technologies involved. Therefore, the threats are considered from a generalised viewpoint to understand the holistic context of CC threats.

To allow researchers to better address CC threats in a structured manner, the analysis of cases is presented as a taxonomy. To better understand the genre and nature of any newly introduced threats, the taxonomy may be applied by categorising them or by considering how threats are related to other categories. Based on the argument made that, threats to CC exist in other computing fields, the proposed taxonomy is applicable to a wide range of issues.

2.4.4 Architectural Weakness for Vulnerabilities

Opportunities exist for threats and vulnerabilities in computer and networking environments and especially in the inter-networking environment (Newman, 2006). This is

carried over to the Cloud context.

One important aspect of security vulnerabilities noted by Kizza (2009) is that the weak points in communication protocols and network infrastructure work as an accelerator for attacks. Poorly configured firewalls and inconsistent security policies may act as accelerators, too (Khalil et al., 2013).

Newman (2006) presents four security weaknesses as part of a networked computing environment: the servers, the client PCs, the Internet and the remote access mechanisms. In addition, the incorporation of distributed systems into the computing context also introduces security concerns. This is due to vulnerabilities in the distributed architecture (Casella, Morin, Harsh & Jegou, 2012).

Mobile platforms and the practice of Bring Your Own Device (BYOD) add further security concerns to those of other computer networks (Fernandes et al., 2014; Xie, Kumar & Agrawal, 2008). The security concerns for BYOD are in user access control and device identification, data protection and monitoring, compliance, mobile application integrity and mobile application security (Eslahi, Naseri, Hashim, Tahir & Saad, 2014).

2.5 Security Models for CC

In this section, existing security models for CC are discussed. The discussion explores different Cloud security models with specific concentration on distributed models or mechanisms. A purely distributed security model for CC could not be found. However, the discussion includes the security models that exist in the CC arena where some of them takes the distributed nature into consideration.

2.5.1 Concept of Security Model in Cloud

In CC, the term security model is not widely used. When it comes to CC security, two different approaches can be found. The first approach defines the arrangement

of Cloud architecture for secured computing and takes a holistic architecture-wide approach; while other approaches take specific security issues (e.g. authentication, identity management, file storage) into account to propose security mechanisms for CC. Some examples of specific security mechanisms are discussed in Section 2.5.2. Examples of Cloud security models include the multi-tenancy model of NIST, the Cloud risk accumulation model by CSA and the Cloud Cube model proposed by the Jericho forum (Che et al., 2011). These models define how the elements should be arranged to form the Cloud architecture for secured deployment and provisioning to work as a reference architecture for the Cloud.

2.5.2 Security Models and Mechanisms for Cloud

Zissis and Lekkas (2012) describe an approach to holistic Cloud security. They recommend the inclusion of a number of trusted third parties to encourage a distributed model of holistic security for Cloud architectures.

Al-Zain, Soh and Parded (2012) describe a security model that enhances CC services that they name as Multi-Cloud Databases (MCDB). The proposed security model addresses the issues of data integrity, data intrusion and service availability. In their proposal, it is suggested that the inclusion of multi-Cloud (integration of several Clouds) to enhance the three issues addressed by the respective security models. In addition, the MCDB model divides the Cloud into three distinct layers, namely, presentation layer, application layer and management layer.

For privacy protection in the Cloud, Zheng, Yang and Chen (2012) propose a probability based noise generation strategy. In their proposal, a random noise request injection is proposed to make it hard for an attacker to distinguish between a real service request and a noise-generated request. The purpose of this approach is to obfuscate and protect the data from being intercepted by malicious third parties.

Mohamed and Abdelkader (2012) propose a new security mechanism for CC that evaluates eight encryption algorithms to choose the strongest one. The encryption techniques taken into account are RC4, RC6, MARS, AES, DES, 3DES, Two-Fish, and Blowfish. The evaluation is implemented by means of a Pseudo Random Number Generator (PRNG). PRNG is used to implement with the encryption algorithm to generate critical data similar to keys and initial vectors. The aim is to use the best encryption algorithm to ensure data security in a cloud.

Vaquero, Roderio-Merino and Moran (2011) discuss an approach, the Trusted Platform Module (TPM), to offer Cloud data-centre security by combining hardware and software. TPM embeds an integrated circuit with basic security features for the software that use it. Systems with TPM are used for creating encrypting cryptographic keys that can only be decrypted by the TPM. The security feature of TPM is often referred to as key binding and key wrapping. However, the concept of TPM is not new and comes with the limitation that it cannot be used in scenarios where simultaneous access by multiple systems is a practicality due to the fact that it requires dedicated hardware (Berger et al., 2006). As simultaneous access by multiple systems is a common part of Cloud architectures, the limitation of TPM is significant within Cloud architecture. As an endeavour to overcome this, IBM introduce virtual TPM which is a virtualised software based TPM capable of spawning multiple instances of TPM (Berger et al., 2006).

On proposing a cyber-security model for a Cloud environment, Rabai et al. (2013) provide justification of economic terms and subsequent minimisation of risk factors present in a Cloud environment. The authors define a strategic approach to mitigate various factors to minimise a threat and associated costs. The cyber-security model proposed by the authors is management focused and deals with factors like risk estimation matrices, mean failure cost, stakeholder security requirements, system components and services, and providers' security concerns.

Chadwick and Fatema (2012) propose a privacy preserving authorisation system for the Cloud. It is an infrastructure service that allows the users to set their own privacy policy to prevent data abuse. The authorisation system ensures that user data and privacy policy are struck together to enforce security. The mapping of policies and resource IDs that are struck together are stored in a place termed as 'sticky store' by the authors.

The Cloudgrid approach proposed by Casola et al (2013) is an amalgamation of Cloud-on-grid and grid-on-Cloud. Cloud-on-grid refers to the Cloud environment built on stable grid infrastructure whereas grid-on-Cloud means using Cloud IaaS approach to build and manage a flexible grid system (Foster et al., 2006). The Cloudgrid approach does so by implementing a Cloud on top of a grid. This is an architecture-centric approach to provide a secured CC environment which consists of three layers: virtual grid, Cloud and physical layer. It applies a security mechanism by introducing third party authentication authorities. The Cloud grid approach can be thought of as a partially distributed approach as it is based on grid computing and incorporates third-party authentication authorities, but the security mechanism itself within the Cloudgrid does not specify or address the distributed nature of a security mechanism.

Arshad, Townsend and Xu (2013) discuss a novel intrusion severity analysis approach which focuses on the security issues that may arise from virtual resource sharing on the physical infrastructure of a Cloud platform. It uses a detection engine and maps known attacks from an attack database to check and detect attacks. A "system call handler" executes the system calls originated by the VMs which, in turn, works in a collaborative manner with the attack database, intrusion detection system and a module for intrusion severity analysis.

Lo, Huang and Ku (2008) propose an intrusion detection system framework for CC. This is a cooperative approach where individual Network-based Intrusion Detection System (NIDS) are deployed in different clouds. Upon intrusion detection in one cloud, the NIDS module in the respective Cloud updates servers in other Cloud about the

detection and adds it to a block list. The Cloud servers operate in a collaborative manner in this approach, and to some extent this cooperative agent-based approach can be thought of as a distributed security mechanism for the Cloud.

Fernandez and Monge (2014) propose a security reference architecture (SRA) for Cloud systems. It uses security patterns which maintains a security catalogue, misuse pattern which contains vulnerabilities or threats, and uses security best practices which contains defences against the threats. The mapping of these databases are then used to define the level of integrity of a Cloud system.

Ghebghoub, Boussaid and Oukid (2014) propose a security model-based encryption to protect data within a Cloud environment. It is based on the Organization Role Based Access Control (ORBAC) model and encryption. The proposed model is based on encryption to provide security for data stored in the Cloud. It uses Cipher-text Policy Attribute-based Encryption (CP-ABE) which is conceptually similar to traditional access control methods (Ghebghoub et al., 2014). It uses secret key to encrypt and decrypt data to ensure proper authorisation.

An architecture-based security model is proposed where the inclusion of separate Cloud to solely serve as a 'security Cloud' is recommended (Srivastava et al., 2011). The proactive security model implies that the distinctive security Cloud will have management server in the security Cloud and a corresponding agent in the public Clouds. If a public Cloud do not have the agent of the security cloud, it is to be treated as non-compatible public Cloud which will be denied access to the private Cloud protected by the security Cloud in the model.

Khamdja, Adi and Logrippo (2013) discuss a security approach for a flexible access control model for CC. The proponents call the approach Category Based Access Control (CatBAC). The model works on a top-down hierarchy where CatBAC meta-model is used and refined by the Cloud providers to build the abstract level access model. The abstract level model in turn is used to build concrete access model for different

organizational sites. This approach is claimed to bring flexibility in access control. Proponents claim CatBAC can also be offered as a Cloud service.

The data security model based on multi-dimension proposed by Xin, Song-qing and Nai-Wen (2012) is based on a three-layer defence architecture. This multi-dimensional model deals with data security in a Cloud architecture. User authentication and data encryption in a layered multi-dimensional architecture is at the centre of this model.

Another security mechanism for the Cloud is the access control model for a CC named AC3 (Younis, Kifayat & Merabti, 2014). It proposes an access control model for Cloud based on Mandatory Access Control (MAC) and Role Based Access Control (RBAC) but it finds the gaps in these traditional access control which are claimed to be mitigated in the proposed access control model. It uses security tags against the threats of security breaches which consists of user role, classification, permission, time, location and random number.

2.5.3 Existing Security Models at a glance

Some of the security models or mechanisms in the previous section are summarised in Table 2.3 on page 71.

Name of the model: The name given to the model by the proponents by which it may be referred to. A security model or mechanism may have a name given by its proponents.

Distributed Security: Whether the security model is distributed or not.

Focus Area: The major functionality of the Cloud that is taken into account. For example, a security model or mechanism may deal only with authentication, or only with inter VM communication and so on.

Architecture-centric or Service-centric: Architecture-centric means the security model or mechanism deals with an architecture-wide context. Service-centric means the

security model or mechanism deals with any specific service (e.g. authentication).

Table 2.3: Summary of the Security Models or Mechanisms

Author(s)	Name of the Model	Distributed Security?	Focus Area	Architecture centric or Service-centric?
Casola et al. (2013)	Cloudgrid	Partially, as it includes third party authentication authorities.	IaaS on grid for virtual resource sharing	Architecture-centric
Arshad, Townsend & Xu. (2013)	Intrusion severity analysis approach	Not addressed	Virtual resource sharing on physical platform	Service-centric
Chadwick & Fatema (2011)	Privacy preserving authorisation system for Cloud	Not addressed	Resource authorisation	Service-centric
Zheng, Yang & Chen (2011)	Probability based noise generation strategy	Not addressed	Obfuscation to protect the data by injecting noise data	Service-centric
Mohamed & Abdelkader (2012)	Data security model for CC	Not addressed	Evaluation of recommendation of existing encryption algorithm by using Pseudo Random Number Generator (PRNG)	Service-centric
Al-Zain, Soh & Parded (2012)	Multi-Cloud Databases (MCDB)	Not addressed	Data integrity, data intrusion and service availability	Architecture-centric
Vaquero, Rodero-Merino & Moran (2011)	Trusted Platform Module (TPM)	Not addressed	Embedding hardware based unique identification for Cloud security	Architecture-centric
Rabai et al. (2013)	Cyber-security model in CC environment	Not addressed	Cloud management perspective to identify risk factors and mitigate those	Architecture-centric

Continued on next page

Table 2.3 – Continued from previous page

Author(s)	Name of the Model	Distributed Security?	Focus Area	Architecture centric or Service-centric?
Lo, Huang & Ku (2008)	Cooperative intrusion detection system	Yes, in terms of updating intrusion detection among Cloud servers	Intrusion detection	Service-centric
Fernandez & Monge (2014)	Security Reference Architecture (SRA) for Cloud systems	Not addressed	Secure Cloud architecture	Architecture-centric
Ghebghoub, Boussaid & Oukid (2014)	Security Model Based Encryption	Not addressed	Access control for Cloud data security	Service-centric
Srivastava et al. (2011)	Proactive model for security in CC	Not addressed	Augmenting organizational security policy	Architecture-centric
Khamdja, Adi & Logrippo (2013)	CatBAC	Not addressed	Access control	Service-centric
Xin, Song-qing & Nai-Wen (2012)	data security model based on multi-dimension	Not addressed	User authentication and data encryption	Service-centric
Younis, Kifayat & Merabti (2014)	AC3	Not addressed	Access Control	Service-centric

As Table 2.1 on page 47 suggests, the distributed nature of security models or mechanisms has not been given much attention. Some of the other security model examples are the data storage security model (H. B. Patel et al., 2012), virtualisation security framework (Park, 2012) and trust evaluation model for CC (Wu et al., 2013). All these service-centric security mechanisms deal with Cloud security but the distributed aspect of a security model is not dealt with. In the next section, the argument that CC requires a new security model is presented leading to a distributed security model for CC.

Distributing a security mechanism for CC has not specifically and distinctly been given attention though some of the security mechanisms exhibit a distributed nature. In the next section, the implications of having a distributed security model are addressed. The implications help to establish the claim of the value of investigating a distributed security model for CC.

2.6 Distributing Security Model: Implications

In this section, the importance and implications of having a distributed security model for CC are presented. Arguments of different authors are taken into account to discuss the distributed nature of CC and how this distributed nature impacts the security within CC. It is also shown how a distributed security model is likely to be more suitable over conventional security models for distributed computing architecture like CC.

CC uses the Internet as the major means to provide access to its services (Grobauer et al., 2011). The Internet uses packet-based data transmission. The packets travel through different paths in a network and are reassembled at the receiving end. The paths through which the packets travel are determined dynamically and often ‘on-the-fly’. Thus, successful functioning of the Internet requires a strong trust relationship among the transmitting elements (Kizza, 2009). The future evolution of computer networks

is termed Internet of Things (IoT), which also relies on the existing packet-switched Internet Protocol (IP) (Raza, Duquennoy, Hoglund, Roedig & Voigt, 2012). On the Internet, data traverses multiple distributed networks to reach its destination. The elements of a Cloud architecture might be distributed where centralized management for security may not be able to cope with monitoring distributed resources. If the security mechanisms are distributed and subsequently there is no single point that results in vulnerability and failure, then a security architecture and mechanism of distributed nature is required. Deibert (2012) notes that CC introduces dynamic cross-border data security issues and thus recommends distribution of security for cyberspace.

CC uses layered architecture objects (e.g. VMs, APIs, applications and services). Within the architecture, the security of a higher layer depends on its corresponding lower layer which complicates the security of CC. Hence the security controls in the Cloud are heterogeneous and complex to manage (Bouayad et al., 2012). Behl and Behl (2012) assert that CC requires a holistic security wrapper to provide a multi-layer security solution. For the layered distribution of resources and their security, the argument is made that a distributed model over a centralised one is credible and consistent with the objectives outlined.

The role of a security mechanism is to diminish or minimise the effects of attacks (Turpe, 2009). If the cause is distributed, the effect may be distributed; and to counter a distributed cause, a distributed security mechanism may be more suitable. Ghebghoub, Boussaid and Oukid (2014) discuss some security models which are partially distributed, showing that distributed security mechanisms ensure enhanced integrity and robustness over security concerns in CC.

A distributed approach enhances data security. The context of CC implies that a distributed approach for data security helps facilitate secret key sharing as well as integrity among multiple service providers (AlZain et al., 2012). This may ensure integrity of a Cloud architecture. With the dispersed nature of CC, a number of

distributed applications are in demand (Cascella et al., 2012). For integrity and security, the nature of the security models for a distributed computing environment like CC should be ascertained. Ceesay, Chandrasekaran and Simpson (2010) argue that sectors like defence, finance and health-care are distributed in nature where information sharing and maintaining privacy is crucial. The authors also state that distributed computing environments are vulnerable to attacks. Taking the distributed social setting into consideration along with the notion that CC is a form of distributed computing, it may not be adequate to have a single point of security management or security mechanism; a distributed security mechanism or authenticity verification mechanism is more credible for a setting which processes and stores information in a distributed manner. This subsequently yields the requirement to have a distributed security model in place. This leaves the impression that conventional non-distributed security models and mechanisms are not adequate for CC. A model for CC may be more consistent if it is distributed and thus does not have a single point of failure (Poh et al., 2013). The complex and centralized management aspects of CC makes it alluring for attackers (Yan et al., 2011) but distributing the security mechanism makes it harder to attack.

In proposing a security model for CC, Park (2012) argues that what make security a crucial concern within CC is the physical or virtual heterogeneous IT resource sharing over a distributed network. Chirag et al. (2013) argue that it is the distributed resources that make the development of a Cloud security model challenging. On the contrary, argument exists that distributed Cloud servers across locations make an architecture more secure (Kumar et al., 2011).

The example of a covert channel attack as described by Vaquero et al. (2011) can be taken into account to highlight the need for a distributed security model. As the covert channel attack is facilitated during inter-process communication to which virtualised Cloud settings are vulnerable, a centralized security mechanism may not be enough to handle the challenge to distinguish between the service requests on different

layers of Cloud virtualisation. If a distributed security mechanism exists to deal with and to incorporate different layers of a Cloud architecture, the nodes (physical and virtual) may be able to work in a collaborative manner to retain integrity and thus security. Grtizalis, Mitchell and Thuraisingham (2014b) suggest that it requires new and innovative approaches to address security concerns in CC.

2.7 Implications of Redundancy in CC

Existing literature suggest that the nature of computing approaches in CC demand the implementation of redundancy for a number of reasons, enhanced security and service availability being some of them. The first reason is to ensure dispersed distribution of Cloud resources to avoid single repository and subsequently, a single point of failure. It is discussed that a redundancy based approach enhances security for CC (Dai, Zhao, Zhang, Qiu & Tao, 2015). The outage of Cloud services are also another concern where redundancy in Cloud can help manage businesses better. Such an example of Cloud outage is the incident of Amazon AWS, which is suggested to have not happened, had there been Cloud redundancy in place. Pamies-Juarez, Garcia-Lopez, Sanchez-Artigas and Herrera (2011) argue that distributed redundant approach in CC can ensure better availability of Cloud resources. In CC, increased availability through redundancy is also discussed by Zhai, Chen, Wolinsky and Ford (2013). From security enhancement to better resource availability, redundancy is taken into consideration in CC. Evidence of such is found in Hernandez-Ramirez, Sosa-Sosa and Lopez-Arevalo (2012) where a number of redundancy techniques for CC storage are discussed.

It is previously discussed that CC is complex due to its distributed nature. Opinions exist that distributed redundancy can help to achieve better resilience in complex systems (Randles, Lamb, Odat & Taleb-Bendiab, 2011). While distributed resources in Cloud may result in performance bottleneck, there exists approaches where redundancy can

help to overcome this issue by using redundancy techniques for latency reduction (Joshi, Soljanin & Wornell, 2015). Lin (2014) proposes a redundancy based Cloud library system that has better structuring and management aspects due to redundancy. Resource availability is thought to be one of the biggest challenges in CC, where redundancy is one of the effective approaches to combat this (Endo et al., 2016).

The above discussion makes it apparent that redundancy might emerge as one of the core criteria for efficient Cloud deployment – both in terms of security enhancement and better resource availability. It also suggests that the performance overheads can be dealt with through redundancy techniques.

2.8 Management Frameworks

Query on how the proposed security model can be developed using existing information and IT management related frameworks or regulations is not unexpected. Thus, to proactively clarify the connection between existing such frameworks and Ki-Ngā-Kōpuku, few frameworks like Information Technology Infrastructure Library (ITIL), Control Objectives for Information and Related Technologies (COBIT), and ISO/IEC 27000 are discussed here. Besides, General Data Protection regulation (GDPR) is also discussed which is a group of laws or regulations based in the EU.

ITIL is a IT service management model (Demont, Breitenbücher, Kopp, Leymann & Wettinger, 2013). GDPR is a regulatory framework that specifies information to be shared, provided, or communicated between entities as part of its operation and compliance (Pandit, O’Sullivan & Lewis, 2018). COBIT is a framework related to IT governance (Bakry & Alfantookh, 2006). ISO/IEC 27000 is a guideline for Information Security Management System (ISMS) to provide control objectives, specific controls, requirements and guidelines, with which an organisation can achieve adequate information security (Disterer, 2013).

Ki-Ngā-Kōpuku is a security model in the form of development framework. It can be complemented with the above frameworks, but cannot be developed using the above frameworks, as Ki-Ngā-Kōpuku and these frameworks serve different purposes. An application developed using Ki-Ngā-Kōpuku will be deployed and operated within the context of any of the above or other similar framework(s).

2.9 Research Questions (RQs)

The discussion and analysis in this chapter suggest that it is important to realise all the facets from which a security breach for the Cloud may emerge. The centralised nature of Cloud resource poses security threat which may subsequently affect Cloud availability. The presented literature review also suggests that the existing threats to the Cloud result in lack of resilience in CC. These findings are the motivation to form the following RQs:

RQ1 What are the contexts from which a Cloud security breach may emerge?

RQ2 What measures can be applied that avoid a single point of failure in Cloud-based systems?

RQ3 How the loss of availability of Cloud services can be minimised?

Section 2.3.1 on page 35 describes how security threats are considered as a major issue in CC. Section 2.3.2 on page 37 illustrates the importance of security for CC. Cloud specific threats are explored in Section 2.4.2 on page 40 that leads to develop a threat taxonomy for CC described in Section 2.4.3 on page 44. Besides, a number of other CC threat related taxonomy are explored and described in this section. The development of the generalised genre-based taxonomy implies that it is important to understand different contexts and factors that may associate security concerns in a CC scenario. These findings lead to formulate RQ1.

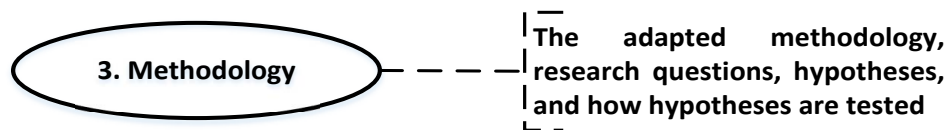
As discussed in Section 2.2.3 on page 33, CC inherits characteristics of distributed computing where assurance of reliability and availability is crucial. Cloud itself is a large distributed network. Section 2.4.3 on page 44 denotes service unavailability and lack of reliability as potential issues in CC. The discussion in Section 2.5.2 on page 65 also reveals that security threats may result in service unavailability and application failure. This may happen if a service has a single point of failure. Section 2.5 on page 64 outlines the existing security models in CC. It is found that the existing approaches to CC security are labelled as confusing by some researchers. It is also argued that the distributed nature of CC resources makes it prone to attack. Furthermore, in Section 2.6 on page 73, the explained implications of a distributed security approach for CC makes it apparent that CC needs further investigation to maximise service availability and to better protect its distributed resources. Researchers' opinion on finding new approach towards CC security warrants further investigation to minimise the scope of single point of failure within a Cloud infrastructure, and to maximise service availability. The above observations lead to formulate RQ2 and RQ3.

2.10 Conclusion

Based on the literature review, it is found that the security approaches in CC required further research. It also suggests that a distributed security model for CC is likely to be more suitable since resources in the Cloud are distributed. The review of existing literature reveals that a truly distributed and decentralised security model for CC does not exist yet, though opinions exist that a distributed security model may be better suited for CC, as CC itself is distributed in nature. These findings lead to design and development of the model proposed. The model then helps to test the hypothesis designed for the research. The next Chapter is 3 where research methodology, hypothesis and the design of the study are described.

Chapter 3

Research Methodology



3.1 Introduction

This chapter presents the design of the study. The discussion addresses the overall research design as well as the research methodology adapted and followed to carry out the research. The RQs are outlined and the hypotheses are formulated to satisfy the RQs. Five methods to test the hypotheses are also discussed. A hypothesis may require more than one method to be tested and thus a mapping of the methods to the hypotheses is also presented. Finally, the adapted research methodology and the research process are discussed. Three major problem areas of CC are identified through literature review that are related to single point of failure, probability of service unavailability result from single point of failure, and lack of resilience as a consequence. The research process is designed to address these identified problem areas in order to develop and propose a

security model for CC.

Section 3.2 of this chapter presents the chosen methodology and the research process. Section 3.3 on page 88 re-visits the RQs as well as hypotheses to explore the RQs. It also outlines the approaches through which the hypotheses are tested.

3.2 Methodology and Research Process

DSR is the base methodology followed for the conducted research. The following discussion explains the Research Methodology and the Research Process.

DSR is supported by, or accepted and followed in numerous research. Few of such examples are Nunamaker, Chen and Purdin (1991), Walls, Widmeyer and Sawy (1992), Walls, Widmeyer and Sawy (2004), Markus, Majchrzak and Gasser (2002), Gregor and Jones (2007), Arazy, Kumar and Shapira (2010), and Agrawal (1993). Vaishnavi and Kuechler (2015) mention a number of resources that shows DSR is accepted and being used as research methodology in leading research contexts. A number of DSR community of practice are mentioned in the discussion, for example, Design Research Society, Association for Information Systems (AIS), AIS Systems Analysis and Design Special Interest Group (SIGSAND), and Informing Science Institute (ISI). A number of DSR centres and research labs are also mentioned, for example, Center for Design Research at Stanford University, IIT Institute of Design at Illinois Institute of Technology, and ISEing—Information Systems Evaluation and Integration Group at Brunel University. To some extent, Ki-Ngā-Kōpuku deals with resilience. Few examples of other resilience-based research that uses DSR are A. Gill, Chew, Kricker and Bird (2016), A. Q. Gill, Phennel, Lane and Phung (2016), and A. Gill, Chew, Bird and Kricker (2015). The research using DSR and the established research centres or labs for DSR not only indicate DSR as an accepted research methodology, but also establish its validity. Besides, specific to the research on Ki-Ngā-Kōpuku, DSR is preferred as it

is an established method for artefact-centric research.

3.2.1 Methodology

As the research methodology, DSR is applied to the study. In information Systems, DSR is an accepted methodology, as DSR aims to “extend the boundaries of human and organizational capabilities by creating new and innovative artefacts” (Hevner, March, Park & Ram, 2004). As discussed in chapter 2 on page 26, it is believed that a purely distributed security model to adequately address the problems of single point of failure, loss of availability and lack of resilience does not exist to date. Development of such security model needs to be focused on its artefacts (e.g. RA and security mechanism). It is for this reason that DSR is best suited to the study, since DSR facilitated artefact-centric research. In the next section the research process within the greater context of DSR is outlined.

3.2.2 Research Process

As DSR is the chosen methodology for the research, a DSR-specific research process is defined. Despite DSR being an accepted approach in IS research, Offerman et al. (2009) argue that the clarity of combining different research methods in DSR is unclear, therefore they propose a DSR research process. An adaptation of the process is illustrated in Figure 3.1 on the next page.

The adapted process combines research approaches suitable for IS research. The process has three major phases: problem identification, solution design and evaluation. Within each phase, several activities or steps take place. The phases are interactive (iterative if required) and not necessarily sequential. The processes may refer back to each other and the execution of the processes produces results of DSR.

The problem is identified at the beginning of the research, which involves Literature

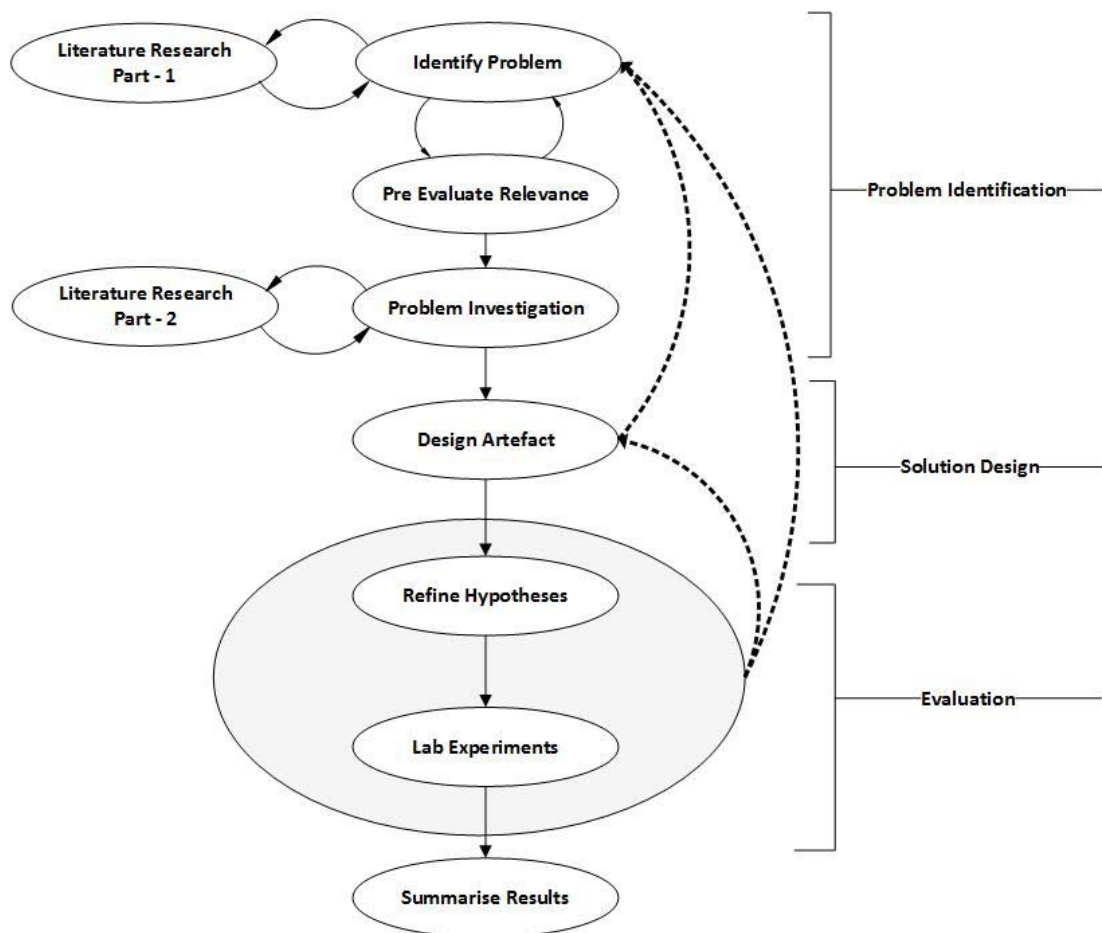


Figure 3.1: Research Process (adapted from Offerman et al., 2009)

Research Part-1. Once the problem is identified, the practical relevance of the problem is established; as Benbasat and Zmud (1999) as well as Rosemann and Vassey (2008) suggest, any problem identified should have relevance to the research domain and context. Forming research questions is part of the problem identification phase and takes part in the Identify Problem step.

The step, Problem Investigation, involves extensive Literature Research Part-2. In Problem Investigation, extensive systematic literature research is conducted to understand the problem domain as well as the degree to which the proposed topic is context-significant. In this step, the areas addressed are: the novelty and contribution of the proposed research, forming hypotheses and determining a research approach and

methods to test hypotheses.

The next step, Design Artefact, is the sole step of the phase Solution Design. There is not much guidance found on artefact design (Offermann et al., 2009), but the MAPS (Matching, Analysis, Projection & Synthesis) proposed by Chow and Jonas (2008) is followed. The conceptual approach of MAPS is taken into account. That is, the Analysis part helps to define the problem situation and the Projection part aids in outlining the artefact to counter the problem identified. The Synthesis part of the approach then helps to define the properties of the artefacts and to formulate the actual artefacts.

The completion of the Solution Design step takes the research to a stage where the Evaluation phase can begin. As the research steps need not be sequential, it is possible to go back to the earlier steps and thus it is iterative if necessary. Upon evaluation, the hypotheses may require refinement. It is ensured that the hypotheses are Mutually Exclusive and Collectively Exhaustive (MECE) (Offermann et al., 2009). Lab experimentation is the final steps of the Evaluation phase. As the research entails employing formal methods to validate the system, the lab experiment does not have conventional meaning within the context of this research. As part of the lab experiment, logical simulation of an example application is carried out. The logical demonstration of an example application is used to establish that an application can be componentised; and with redundant components, an application is able to survive even if one of its component is taken down. The example application development and logical simulation proves the feasibility of the idea of application componentisation and redundant distribution, and thus the feasibility of the security model is evaluated, hence the name of the phase is Evaluation.

Successful completion of all the phases of the research process enables us to summarise and write down the results and outcome of the research, along with a rationale and associated discussion.

3.2.3 DSR Evaluation

This section describes the evaluation approach and tool for DSR used in the conducted research. Formal methods used along with DSR as a validation tool. This section justifies the use of formal methods as a validation tool.

DSR Evaluation Approach

For evaluation within DSR for the research conducted, the strategic DSR evaluation framework proposed by Pries-Heje, Baskerville and Venable (2006) is used. The Artificial Setting in Ex Post is used for evaluation. The use of Ex Post and Artificial setting are validated by researchers, for example, Venable (2006), J. G. Walls et al. (1992), and Purao and Storey (2008). In Ex Post, evaluation takes places after an artefact is constructed (Pries-Heje et al., 2008). In the research presented, validation of the model as presented in Section 7.2 on page 145 is done on the model constructed. This conforms to the requirement of Ex Post being a credible evaluation approach for the research presented. As part of Ex Post, artificial settings are used. In artificial setting, the evaluation of artefacts is not limited to only experimental settings. It may include imaginary or simulated setting that allow to test a technology or its representation under substantial artificial condition (Pries-Heje et al., 2008). The logical simulation of the model as well as using formal methods for model validation as presented in Section 7.2 on page 145 conforms to such approach.

Models and methods are the artefacts produced as part of the research. Models and Methods are mentioned as artefact types by March and Smith (1995), Hevner et al. (2004), and Winter (2008). These are commonly accepted in DSR (Vahidov, 2006; Vom & Buddendick, 2006). Section 5.2 on page 113 and Section 5.2.1 on page 113 describe models and Section 6.2 on page 127 describes methods which are some of the artefacts yielded as the outcome of the research.

Formal Method as a Validation Tool

As part of evaluation, Formal Methods are used as validation tool within the context of DSR. This is described in Section 7.2 on page 145. Part of the validation approach is model checking using Formal Methods. The use of Formal Methods as validation tool are inspired by some research projects that used Formal Methods for validation. For example, Easterbrook et al. (2008) used Formal Methods as validation tool for modelling spacecraft fault protection systems. They mention that Formal Methods provide a number of ways to help in validation. One such way is model checking. Antoni and Ammad (2011) use Formal Method as a validation tool. Kuhn, Chandramouli and Butler (2002) state that despite of the need for human intervention, Formal Methods indeed have a role to play in validation. They state that Formal Methods can be used to validate conceptual models. The research presents conceptual models and their validation using Formal Methods, as described in Section 7.2 on page 145.

3.2.4 Other Research Methods

This section presents other research methods that are explored in addition to DSR, to find the most suited one for the conducted research. As discussed earlier, DSR is chosen due to the research being artefact-centric. Other research methods considered are Phenomenology, Grounded Theory and Standard Quantitative Model. The following discussion addresses why DSR is a more suitable research method over these methods.

Phenomenology

Phenomenology is a qualitative research method (Chamberlain, 2009). This method is used to describe the meaning of the lived experience of a phenomenon (S. N. Khan, 2014). To carry out research, a researcher using this approach needs to rely on the experience of those who encountered the phenomenon of interest. Thus it involves

real life participants where observation is required on the samples (participants). The outcome of the research conducted using Phenomenology is a thematic description of pre-given structure of lived experience (Starks & Trinidad, 2007).

Grounded Theory

In grounded theory approach, theory is discovered by examining concepts grounded in existing data. The goal of this is to develop a social process (Starks & Trinidad, 2007). This approach requires participants who have gone through a specific phenomenon under different conditions (Corbin, 1990). Interview is a research tool within this method where participants describes their experiences. This approach is considered in types of research where the goal is to generate theory from the participants' experience (Bowen, 2006).

Standard Quantitative Model

Quantitative research uses statistical method to analyse gathered data. It emphasises on generalising gathered data across groups of participants, or on explaining a particular phenomenon (Creswell, 2003). In this approach, data is gathered using structured research tools like questionnaire. Data in this approach are in the form the numbers and statistics (Dixon-Woods, Agarwal, Jones, Young & Sutton, 2005). The data is collected from a historical source, or by interviewing the participants and then analysis is carried out on the collected data set to find correlations among variables determined at the research design stage. This approach is used to establish relationship between two variables from a population. Quantitative data is used to interpret social meaning (Mingers, 2001).

The research on the proposed security model is an IS research involving no participants. The research, as described earlier, is artefact-centric and involves developing conceptual models and mechanism as artefacts as well as checking validity of the

developed artefacts. The research on the security model is neither on observing the experience gained from any phenomenon, nor it is about developing any social process. Besides, the conducted research does not aim to explain a particular phenomenon, or generalise gathered numerical data to interpret social meaning. The above research methods are thus not better suited for the conducted research, when compared to DSR.

3.3 Research Questions (RQs) and Hypotheses

As part of the study design, the RQs are formed and the hypotheses to test the RQs are developed. The following discussion presents the RQs and hypotheses. The discussion also address the methods used to test the hypotheses.

3.3.1 Research Questions

Literature review suggests a number of aspects in Cloud security that required further attention. For example, the discussion in sections 2.3 on page 35 and 2.4 on page 38 warrants that factors and contexts of Cloud security breaches needs to be well realised. Sections 2.5 on page 64, 2.6 on page 73 and 2.7 on page 76 reveals that due to distributed nature of Cloud resources, leaving single point of failure in CC architecture makes CC vulnerable to attacks that directly affects the resilience and availability of Cloud resources. These are the basis of motivation for the RQs formulated.

Following are the revisited RQs:

RQ1 What are the contexts from which a Cloud security breach may emerge?

RQ2 What measures can be applied that avoid a single point of failure in Cloud-based systems?

RQ3 How the loss of availability of Cloud services can be minimised?

Answers to the above RQs are sought through primary and secondary research

where primary research includes development and validation of artefacts resulting from the conducted research, and secondary research includes the use of previously published data as the outcome of researches done by other researchers.

To answer RQ1, the concept of CC is taken into account. At the same time, the security aspects of CC, the threats and vulnerabilities of CC are explored. The comparison and contrast of Cloud breaches with other threats found in traditional Computer Networks helps to find the gap that exists due to the introduction of CC. This leads to RQ2 where the requirement and feasibility of a distributed security model for CC is explored. As the Literature Review suggests, it is found that existing approaches to Cloud security are not adequate and new security models/mechanisms are required. In this regard, a distributed security model could be suggested for CC, and such realisation is the motivation behind exploring the answers to RQ3. The research to resolve RQ3 results in developing various logical and physical artefacts of the distributed security model for CC.

3.3.2 Hypotheses & Testing Methods

To satisfy the research questions, forming and testing the hypotheses is required. The outcome of the hypothesis testing will subsequently help to find answers to the research questions. Five methods are also determined to use to test the hypotheses. The hypotheses and the methods to test the hypotheses are discussed below.

Hypotheses

The following four hypotheses are formed at this stage of the research:

H1 Existing software architecture provide the conditions for a single point of failure.

H2 Threats to CC are unique when compared to other computing models.

H3 In a decentralised and distributed architecture, the problem of a single point of failure can be eliminated.

H4 To minimise service unavailability and to improve CC security, the design strategy of a security mechanism is important.

The above hypotheses are tested using a number of testing methods described later in this section. It is found that the security threats in CC include all threats existent in a networked computing scenario. At the same time, a CC scenario adds more Cloud-specific threats to the list. The development and testing of the distributed security model emphasises that a threat can be momentarily discovered and counter-measured if the distributed security model is deployed.

Testing the Hypotheses

To test H1, it is important to establish whether the distributed arrangement of a computing scenario could widen the opportunity or make room for new attacks or threats to emerge. Contemporary trends and approaches indicate that computing is shifting towards a distributed scenario from a centralised scenario (Cardoso & Simões, 2012). If the historical data (i.e. literature review) provide evidence that a non-distributed computing environment is less vulnerable than a distributed one, or the distributed arrangement would open up opportunities for new or unknown breaches; it could be concluded that the distributed nature of a computing environment widens the threat opportunities. This brings the interest related to H1 into focus to explore whether a distributed security mechanism or a centralised security mechanism could be suggested for a distributed computing environment. The disproving of H1 would imply that a distributed security mechanism could potentially hinder the emergence of new threats or minimize the likelihood of existing threats, which would strengthen the motivation of the research towards outlining a distributed security model for CC. Research on H1

explores whether the distributed nature of a security mechanism could be a driving factor to introduce new or unknown threats or vulnerabilities. The hypothesis would be disproved if the distributed nature of a security mechanism is not a source of any kind of threat.

H2 deals with identifying and comparing the security threats in CC to those of other networked computing approaches. This is done to eventually appreciate whether the introduction of CC would introduce added security concerns or not, as well as whether the existing security issues would equally be of concern within the CC context. Proving the hypothesis would confirm that all known or existing security concerns are applicable to CC. Thus proving or disproving H2 would help to set a context for the security concerns that the proposed distributed security model is required to address. A secondary literature review would provide information on the old and new security threats in CC and other networked computing scenario. This in turn would help to compare and contrast the threats.

The question of interest in H3 is whether a distributed security mechanism would be able to recover from an attack at the very early stage. To accomplish this, it is required to create a test bed environment to emulate an attack in the environment where the developed security model would be implemented. This subsequently allows the capability of the security model to be determined and shows whether the distributed security model is capable of identifying the threat at an early stage and a distributed security model helps an application to serve uninterruptedly even in the case of a breach or an attack.

Proving H4 would establish the authority of the distributed security model in providing better security for CC. Secondary research would primarily be used to establish whether a distributed security model is an effective tool for CC security. Combined with the outcome of H3 testing, the outcome of H4 testing would then be able to indicate the possible effectiveness of the distributed security model as a countermeasure

for CC security concerns. Part of this would also be accomplished by re-visiting the threat taxonomy developed to test H2 and by analysing and subsequently predicting the possible performance for the security model for different kinds of threats and attacks. This would eventually help to determine the capability of the security model and the future development that it may require.

Hypothesis Testing Approach

The approaches to test the hypotheses are discussed below. the testing methods are only approaches to test the hypotheses, that in turn sit within the context of the followed research methodology. Following five methods are formulated as the approaches to test the hypotheses:

- M1 Explore historical evidence to understand the implications and limitations of different Cloud security mechanisms.
- M2 Develop the theoretical model to define the distributed security model. The principles of DSR approach will be employed in this method. Formal methods within the context of the principles of DSR approach will be employed in this method.
- M3 Explore approaches of different processing aspects of a distributed security model, and determine best suited algorithm for the processing.
- M4 Explore approaches of how an application can be self-healing and can uninterruptedly continue to serve even in the case of a breach or attack, and determine best suited algorithm for this. Formal methods within the context of the principles of DSR approach will be employed in this approach.
- M5 Explore approaches of how a logical Proof of Concept (PoC) can be developed, and ways to validate various aspects of the distributed and decentralised security

mechanism; and determine the impact and integrity between Cloud architecture and security model. Formal methods and logical reasoning for system validation within the context of DSR will be employed in this method.

Explanation of the Methods

The following describes the methods into further detail.

M1 denotes carrying out a secondary research. The secondary research is carried out in the form of a literature review by consulting publications from outside sources. Peer-reviewed journal articles and conference papers on CC and computer security are explored. To get updates on contemporary trends and breaches in CC, recent newspaper articles are also taken into account. This helps to understand the nature of current threats, to define the problem area and to develop a threat taxonomy for CC. The literature review is done by following a concept-centric approach instead of an author-centric approach as denoted by Webster and Watson (2002).

M2 is the approach to define the high-level conceptual view of the logical structure and specification of the distributed security model. The outcome of this method associated the artefact of high-level conceptual view of the security model that is Turing complete, and the distribution algorithm of the components is shown to be holding the Church-Turing theorem that enforces the credibility of the distribution approach as valid algorithmic approach. Thus, the illustration of the high-level view of the distributed security model is validated by using formal methods. In M2, further elaboration of the distributed security model is done by developing the reference architecture of the model. As discussed in chapter 5 on page 112, the RA for the proposed distributed security model would require customised arrangement of the elements of the Cloud infrastructure for its processing to take place. This arrangement of the RA for the security model is also part of M2.

M3 addresses the processing details of the security model. To illustrate the processing approaches of the security model, a number of algorithms are developed and the steps are shown by means of flowchart.

M4 is a further addition to the processing approaches defined in M3, but with a specific focus on finding how a distributed and decentralised security model can make an application self-healing in order for it to continue functioning even in the case of an attack or breach. M4 involves the development of the algorithms that would prevent service unavailability in the case a Cloud Server is compromised.

M5 deals with developing the logical PoC of the decentralised, distributed security model. As part of M5, an example application is taken into account and logically deployed within Ki-Ngā-Kōpuku architecture. The validation of the model and security mechanism is done using formal methods to establish the computational feasibility of the security model and associated mechanism. Logical simulation on part of the model is also considered under this approach.

RQs, Hypotheses & Methods: Mapping

Methods are formulated to test the hypotheses, and the hypotheses are developed to seek answers to the research questions. Thus, there is a clear and direct relationship among these three. A mapping is presented in this section to illustrate the relationship and dependencies among the RQs, hypotheses and the methods to test the hypotheses that are discussed earlier.

Figure 3.2 on the next page depicts a mapping among research question to hypotheses to methods. As illustrated in Figure 3.2 on the following page, three RQs are explored by proving and disproving four hypotheses which in turn are achieved by using five methods. Figure 3.2 on the next page also illustrates which hypotheses are related to which RQs, and which method(s) is used to test which hypothesis. It also illustrates the two adapted methodological approaches followed for the methods undertaken, namely

DSR and Formal Methods. The following discussion discusses the methodology and research process.

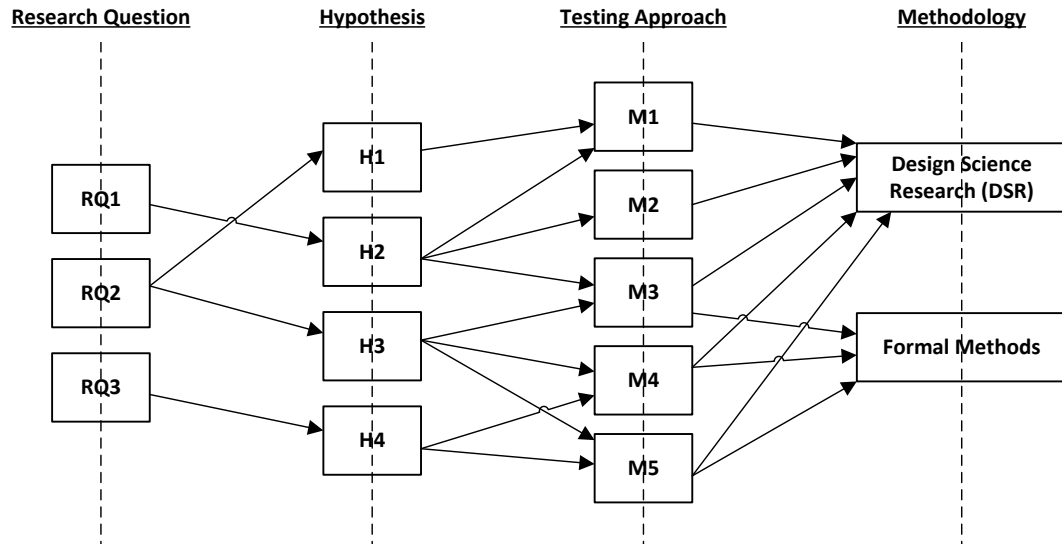


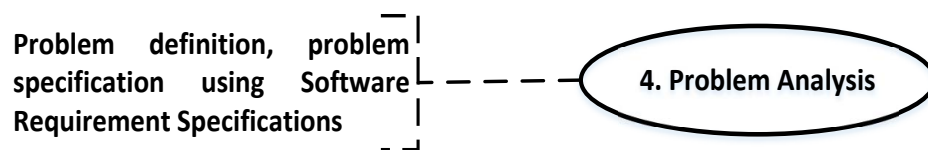
Figure 3.2: Mapping among RQs, hypotheses and hypotheses testing methods

3.4 Conclusion

The research process is an adaptive one. As the research progresses, a number of changes are introduced to the initially ascertained criteria. For example, the RQs are reviewed and some amendments are done. This is not beyond expectation as it is anticipated that iteration to the earlier stages of the research process may be required. The selection of Research Methodology and determination of Research Process upon developing RQs, Hypotheses and testing methods helps to initiate the research in a structured and planned manner. The research design helps to determine the next step at any given point of the research. Based on this, the research problem needs to be defined at this stage. Also, the high-level specifications and requirements of the proposed security model need to be set. These are addressed in the next chapter.

Chapter 4

Problem Analysis



4.1 Introduction

In earlier discussion, it was established that CC security warrants ongoing research. With this as motivation, the research problem is defined in this chapter. First the problem that is solved by proposing the distributed security model is defined. The problem is defined in Section 4.2 on the following page and specifications of the problem are discussed in Section 4.3 on page 98 below. In Problem Definition, the problem that the conducted research aim to solve is discussed. Problem Specification further explores the expectations from the outcome of the research. In this chapter, the expected features/aspects of the proposed security model is discussed.

4.2 Problem Definition

The literature suggests that a distributed approach to security is well suited to CC. If a distributed approach to manage Cloud resources can be deployed, it would have more credibility and integrity in terms of Cloud security.

Literature review also suggests that a single point of failure leaves Cloud architecture vulnerable to service unavailability and subsequently lack of resilience. Resource availability may become an issue in the Cloud for which redundancy is thought to be one of the remedial approaches.

The above problems are addressed in the research. The research intends to develop a distributed and decentralised security model for CC. The distributed security model would divide an application into several parts. Let these parts be called “components” of the respective application. These components would reside in different Cloud servers in a random manner. Such random distribution would make the security model a distributed one.

4.2.1 Design Definition

The proposed security model would be decentralised with redundancy, based on its characteristics and the fact that the security model would not have any single and centralised core. An indefinite number of components may be part of the distributed model (and in turn the application deployed using any architecture built upon the principles of the security model), and the distributed components would reside in different Cloud servers. Thus the security model would be both distributed and decentralised. It would be distributed by having the components in different Cloud servers, and it would be decentralised by not having a single “core” of the applications deployed using the model.

Apart from the above, the components would be replicated unpredictably. This

implies that an application will not only be made up of several components, but also the components of the respective applications will have multiple replicas scattered in different Cloud servers. This aspect - while creating a random scenario of the components, making it harder to predict the overall picture of the security model – eliminates a single point of failure, as the system has no single core and the components of the system are scattered among random numbers of Cloud servers.

To sum up, an architecture for distributed security in the Cloud is defined and proposed. Within the architecture, an application will be divided into a number of components and will be distributed among different Cloud servers. All the components distributed among the servers will work collaboratively to accomplish the intended functionality of the componentised application, but no single core or centralised management would exist for the application. The components of the application will have their redundant copies, that is, multiple copies of themselves. Thus, major characteristics of the security model are that it is distributed, decentralised, has no single point of failure and associates redundancy. The specific aspects and features of the distributed security model are outlined in the next section.

4.3 Design Specification

In this section, the specifications of the distributed security model are presented in the form of SRS. The specification illustrates the overview of the security model to be developed. It also specifies the functional and non-functional requirements. Five functional requirements are identified and discussed in SRS. Factors like data requirements and system constraints are also taken into account in the SRS. Finally, the application-framework-perspective SRS is presented and discussed. Figure 4.1 on the following page illustrates the elements discussed in this section.

The justification on specifying SRS is discussed below.

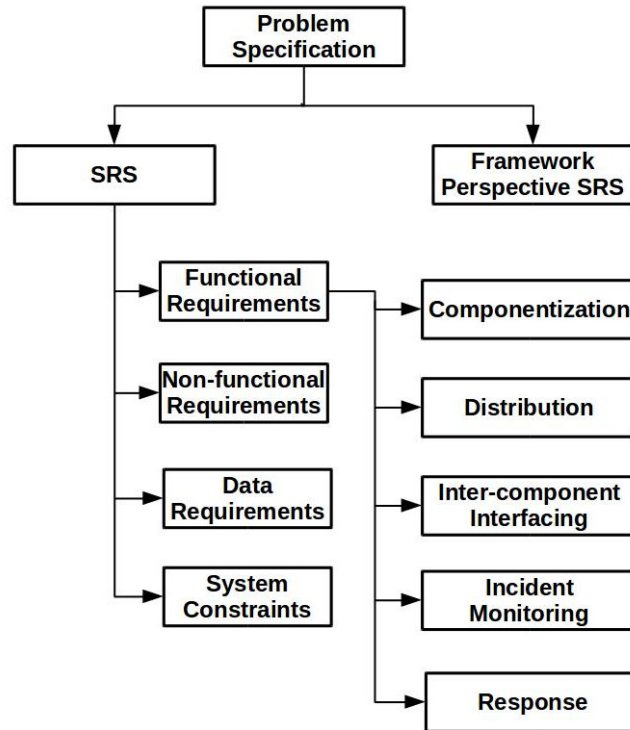


Figure 4.1: Problem Specification

4.3.1 Software Requirements Specification

This section is the SRS for the distributed security model. Capabilities of a software are defined in SRS (W. Wilson, 1999) which is written from the developers' viewpoint (Lauesen, 2002). According to Dogra, Kaur and Kaushi (2009), SRS serves as the base document for software architecture specifications. The SRS document works as the baseline to outline architectural decisions including a system's structures, components and their relations, attributes and characteristics (Gross & Doerr, 2012). Thus, SRS defines the high-level requirements of a proposed system where detailed processing or how the required functionalities will be actualised are out of the scope of an SRS. An overview of the security model is discussed in this section. The functional, non-functional and data requirements form the major part of this SRS, followed by system constraints. Besides, the application framework perspective of the requirements is also

outlined. The detailed processing steps are discussed in Chapter 6 on page 125.

Overview of the System

Ki-Ngā-Kōpuku defines an approach to secure Cloud infrastructure. It is an application development architecture for secure computing within a CC environment. The blueprint of Ki-Ngā-Kōpuku depicts the steps of how an application can be developed and deployed in a distributed and decentralised manner without having a single point of failure. It aims to componentise an application into several components and then distribute the components among different Cloud servers in a random manner. The components of an application would have redundant copies, implying that any given component can be found in a random number of Cloud servers. As a result, the components would be distributed among a random number of servers and the number of copies of a component as well as the list of specific components in any given Cloud server would be unknown. This subsequently would create a random scenario where the components are distributed in a random manner, to make Ki-Ngā-Kōpuku a purely decentralised architecture with no single point of failure, since there is no system core. The practical application of the security model is in the form of an application development architecture that can be used by the CSPs or Cloud application developers to develop and deploy Cloud applications. Thus, the architecture of the security model defines how an application can be componentised, and how the components would interface among themselves to communicate. It also defines the process of monitoring for errors or exceptions while an application is operational.

The system is self-healing. Ki-Ngā-Kōpuku employs redundancy by replicating its components among all the servers in a random manner. There is no system core due to the fact that the components together make the whole system and the components are randomly scattered among a random number of Cloud servers. Thus, if any server or a component residing in that respective server gets compromised or acts maliciously,

the server can be forcefully taken down to exclude the malicious part from the system without losing any part of the whole system. This aspect makes Ki-Ngā-Kōpuku self-healing with the simple approach of ‘cutting-off’ the malicious part, since the redundant copies are existent elsewhere in the architecture to work as substitutes.

The life cycle or the deployment steps of an application using Ki-Ngā-Kōpuku architecture is made up of the following steps:

1. Componentisation
2. Distribution
3. Inter-component Interfacing
4. Incident Monitoring
5. Response

The detailed explanation of the above are discussed in Chapter 5 on page 112, as the SRS forms the baseline for further exploration of the details of system architecture.

The followings are the key aspects and facts about Ki-Ngā-Kōpuku:

- It is an application development architecture.
- It defines componentisation, i.e. how the componentisation would take place.
- It defines component distribution, i.e. the strategic approach to divide an application into several components.
- It defines interface among the components, i.e. how the scattered components of an application would communicate among themselves.
- It defines exception monitoring, i.e. what to do if a component looks suspicious, or if there is an indication of a possible initiation/endeavour of a threat or a breach.

System Requirements

Requirements are categorised into functional requirements, non-functional requirements and so on (Kaiya, Sato, Osada, Kitazawa & Kaijiri, 2008). Paech, Dutoit, Kerkow and

Knethen (2002) argue that any architectural decision must be tightly integrated with the functional and non-functional requirements. Since the proposed system will provide an architecture for secure CC practices, it is important to determine these requirements.

Functional requirements specify the intended behaviour of the system that can be expressed as functions or tasks the system is expected to perform (Malan & Bredemeyer, 2001). Non-functional requirements are the expected quality of a software or system, also referred to as quality requirements (Ameller, Ayala, Cabot & Franch, 2012). Functional and non-functional requirements are addressed in this SRS. The discussion on data requirements is also part of the SRS which defines the type of data that the system is expected to deal with. The system constraints are also outlined in the SRS. The constraints of a system or software draws the extent of the degree of freedom and thus are the limiting factors for the respective system or software (Keeling, 2014).

The above-mentioned requirements and constraints are described below.

1. **Functional Requirements:** Functional requirements are divided into five sub-categories as discussed below:

(a) **Componentisation:** As part of componentisation, the system will carry out the following functions:

- i. Choose the application that will be deployed within Ki-Ngā-Kōpuku architecture.
- ii. Determine the number of components the application will be chopped down into.
- iii. Separate the application's source code into the number corresponding to the number of components determined.
- iv. Create containers for the components.
- v. Add functionality for different components.

- vi. Add functionality for component distribution, inter-component interfacing, incident detection and interfacing.

Componentisation is the prerequisite step for system deployment. First, it is required to decide on the number of components into which the application to be deployed will be chopped down. The componentisation is a manual process to be accomplished by the developer(s) to prepare the system to be distributed. The number of components must be greater than one. It is required to decide the total functionality, database requirements and total number of requests that could be initiated by the end-users while using the application. Subsequently, the functionality and part of database that will be held in each component needs to be decided based on which functionalities will be embedded into the components. As Ki-Ngā-Kōpuku is a security model in the form of an architecture, it can be applied to develop any application, and the functionalities, database requirements and end-users' requests will depend on the application being developed using Ki-Ngā-Kōpuku architecture. Each component will have its own capability to be distributed, to interface with other components, to detect incidents and generate proper response to incidents. These functions are added once the base component is developed and are discussed below.

The specific steps and processing details for componentisation are subsequently developed based on the above, and explained in Chapter 6 on page 125.

- (b) **Distribution:** As part of Distribution, the system will carry out the following functions by employing distribution algorithm:
 - i. Create an initial scenario for Ki-Ngā-Kōpuku to start working.
 - ii. Distribute components to the newly added server.

Once the components are ready, they are ‘planted’ in a server, the first server in the scenario. The components then ‘scan’ for other servers present in the Cloud infrastructure and distributes the components among all other servers. The distribution takes place whenever a new server becomes ‘live’ within the infrastructure. The distribution of components need to be nearly balanced. For example, if there are two components ‘a’ and ‘b’, the total number of ‘a’s and ‘b’s must not differ by more than one. Thus, if there are five copies of ‘a’, the number of copies for ‘b’ must be either four, five, or six. If a server goes down and the total number of components gets imbalanced, a distribution will take place to balance it to adhere to the above.

The specific steps and processing details for distribution are subsequently developed based on the above, and explained in Chapter 6 on page 125.

(c) **Inter-component Interfacing:** As part of Inter-component Interfacing, the system will carry out the following functions:

- i. Create an interface among the components distributed in the servers.
- ii. Define how the components communicate among themselves.
- iii. Decide which servers will process any specific request.
- iv. Decide which chosen server will send processing outcomes back to the user or requestor, which would incorporate randomisation in choosing servers to ensure any specific server(s) is/are not chosen successively.
- v. Have a mechanism to communicate with the right nodes.

All the components will have functionalities to interface among themselves. The specific steps and processing details for inter-component interfacing are subsequently developed based on the above, and explained in Chapter 6 on page 125.

(d) **Incident Detection:** As part of Incident Detection, the system will carry

out the following functions:

- i. Continuously monitor key indicators and at any given time, decide if unexpected scenarios tend to arise/occur.
- ii. Locate the component(s) and server responsible for attempting to make infrastructure scenarios inconsistent.
- iii. Make the incident an architecture-wide knowledge, i.e. broadcast the detection to all other servers.

The specific steps and processing details for incident monitoring are subsequently developed based on the above, and explained in Chapter 6 on page 125.

(e) **Response:** Upon incident detection, the system will carry out the following functions:

- i. Either quarantine or shut-down or exclude the concerned component(s) and the server. However, as the system will have redundancy, taking one server down would never affect the functioning of the system. Thus, excluding and shutting down an affected server is the safest options. Some purpose built hidden nodes can be used for this sole purpose; these hidden nodes are the servers that would exist in the architecture but are not visible to all the nodes. Upon incident detection, these hidden nodes will come into action to shut the affected server down, while other servers will exclude the affected server from their trusted list.
- ii. Make the response decision taken an architecture-wide knowledge. The response decision would be based on some specific rules and thus the response decision taken by all the servers involved must automatically be the same.

- iii. Create an incident report or generate an exception flag and log it or send outside the architecture for human intervention. This could be done by a number of means ranging from sending an SOS signal to a remote computer(s) or emailing the report to the system administrator. This is required despite the system being self-healing.

The specific steps and processing details for response are subsequently developed based on the above, and explained in Chapter 6 on page 125.

2. **Non-functional Requirements:** The non-functional or quality attributes are specific to the software/application of interest and thus a generic ‘common’ specification on the quality attributes for software in general would be impractical. The quality attributes will depend on the application developed.

However, general expectations on the core functionalities of Ki-Ngā-Kōpuku are expected to be executed according to the processing blueprint of the architecture. In the case of unexpected errors in processing, the system should be able to continue without that specific task being accomplished. For example, if a new server is added but component distribution cannot be accomplished, Ki-Nga-Kōpuku should be able to continue its operation without the new server, due to its characteristics of redundancy with randomly distributed components among the servers.

3. **Data Requirements:** Since the distributed security model is an application architecture, no specific data requirements are defined. As a reference framework, the architectural and processing blueprint defined by Ki-Ngā-Kōpuku may be used to deploy any software system which will be able to protect itself with self-healing capability. It will then stand with the software or tool developed using Ki-Ngā-Kōpuku for the data used within the context of that software. Thus, the data requirement for an application will depend on the type of application

developed using Ki-Ngā-Kōpuku. The way Ki-Ngā-Kōpuku works has no conflicting mechanism with data types. Any type of data (e.g. text, image, audio, video) can be processed by an application developed using Ki-Ngā-Kōpuku.

4. **System Constraints:** The technical system constraints for Ki-Ngā-Kōpuku are discussed in this section. As explained by Keeling (2014), “technical constraints are fixed technical design decisions that absolutely cannot be changed...Sometimes a team may choose to create a constraint, perhaps to simplify the world in which they are working.” The following constraints are noted for the system:

- (a) The number of components is N where $N > 1$. The number of existing components must be nearly balanced, where nearly balanced implies that the number of any specific components at any given time should not exceed by more than one compared to the number of other components. For example, if there are five copies of a component, another component should have no less than four and no more than six copies. The bandwidth and processing requirement for any application developed using Ki-Ngā-Kōpuku will be determined by the bandwidth and processing requirement of the respective application. The resource requirement of the system will tend to be higher as the number of servers grows. The more the servers allocated for Ki-Ngā-Kōpuku, the more the processing requirements will be. It is up to the developers to decide how many servers will be employed to deploy an application using Ki-Ngā-Kōpuku. The nature of the application, performance expectations and data requirements will be the driving factors for deciding the above.
- (b) The number of potential nodes is unlimited. However, there is a lower limit on the number of nodes that should exist at any time. The number of nodes/servers must be greater than N , if there are N components. For

example, if an application is componentised to have three components, there must be a minimum of four nodes in the architecture.

- (c) The initiation of the system is manual. That is, the componentisation and implementation of the initial components are done manually and is not an automated process. Future research would involve automating the initiation. The system initiation phase is not done via remote access. Remote access for system initiation is considered as future research for Ki-Ngā-Kōpuku and thus is left out of scope of the research presented. As well, when a malicious server is force shut down, the deletion or wiping out of the server is currently a manual process. Again, automation of the above is listed as further research and future development of Ki-Ngā-Kōpuku.

4.3.2 Framework Perspective SRS

As stated by Pree (1995), an application framework needs to define frozen spots and hot spots within the framework – where the frozen spots are the parts of the framework that cannot be changed, and the hot spots are the parts of the framework where new, ad-hoc or customised functionalities can be added by the programmers/developers. Figure 4.2 illustrates the concept of frozen spots and hot spots of an application framework, and the following discussion defines the frozen spots and hot spots of Ki-Ngā-Kōpuku.

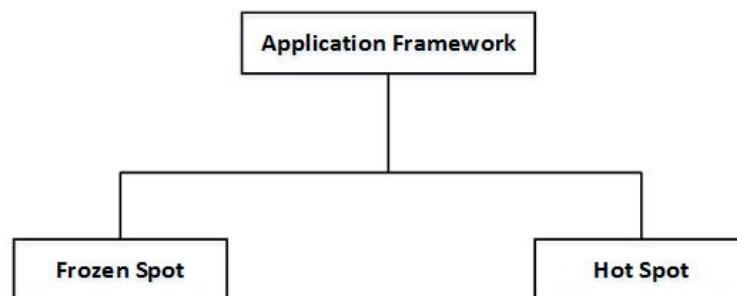


Figure 4.2: Frozen and Hot Spot [W. Pree (1995)]

As described earlier, the core functionalities that define Ki-Ngā-Kōpuku are Componentisation, Distribution, Inter-component Interfacing, Incident Monitoring and Response. Regardless of the application being developed using Ki-Ngā-Kōpuku, these would remain same. Thus, the above functionalities are the frozen spots for Ki-Ngā-Kōpuku. On the other hand, the functions to be added to components of an application as well as the data requirements would depend purely on the respective application and this would vary from one application to another. Thus, the application-specific functionalities would be contained in the components, and the data requirements fall in the hot spots of the framework.

4.4 Types of Security Provided

Ki-Ngā-Kōpuku does not provide solutions to all kinds of security concerns and threats, but addresses three security concerns. The security concerns addressed by Ki-Ngā-Kōpuku are discussed:

Application Security: Ki-Ngā-Kōpuku secures an application from being taken down and thus increases service availability time. Due to having its components scattered among Cloud servers randomly, an application cannot be taken down. All the components of an application may not reside in any given server. At the same time, all the components have their replicas distributed among the Cloud servers. Thus, if a server is taken down forcefully, the application will still continue to function without any interruption. Thus, Ki-Ngā-Kōpuku is decentralised, distributed, and has no single point of failure. Also, if designed with proper strategy, an application within Ki-Ngā-Kōpuku is able to battle with significant resistance against DoS or DDoS attacks. This can be achieved by deploying nodes among a number of different computing architectures located in different places. Besides, due to the approaches that would be employed by the security

mechanism of the model, man-in-the-middle attacks would become powerless and imitating a single node would not benefit the attacker. As discussed below, due to context illiteracy, it would be challenging for an attacker to imitate all the nodes.

Context Illiteracy: At any given point in time, an attacker should not be able to recreate the software architecture built on Ki-Ngā-Kōpuku and thus may not be able to launch an architecture-wide attack. Ki-Ngā-Kōpuku ensures illiteracy of its context to not providing any opportunity to learn about its overall context.

Self-healing: Ki-Ngā-Kōpuku is self-healing. As mentioned above, taking any server down forcefully will not result in taking an application down. Applications deployed within Ki-Ngā-Kōpuku architecture are able to have their components self-replicated and create redundant copies of the components. Thus, if a component is compromised, Ki-Ngā-Kōpuku will simply discard malicious components, and create a copy from another replica of the same component. This replication ensures that application developed within Ki-Ngā-Kōpuku architecture have self-healing capability. The primary objective is resilience and this is served by self-healing. The above by definition is resilience, and this can be qualified to be called self-healing in the entirety of the whole system when a compromised node is replaced by another node; as replacing a malicious node would heal the whole system.

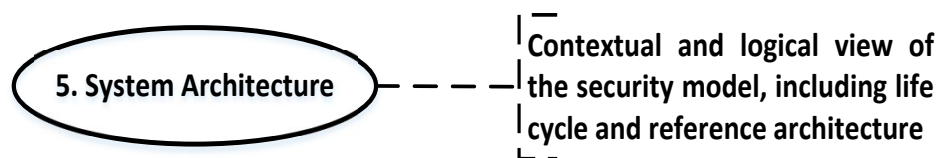
4.5 Conclusion

This chapter outlines the big picture of the proposed security model. The discussion of various requirements and constraints defines the scope of the security model. The discussion forms the basis for detailed and low-level design aspects and specifications

for the security model. Besides, the security aspects addressed by the security model are also discussed. The system architecture designed based on the identified requirements, . The system architecture incorporates the processing details to completely portray the picture of Ki-Ngā-Kōpuku. Architecture for the security model is discussed in the next chapter.

Chapter 5

System Architecture



5.1 Introduction

As argued earlier, a security model consists of an RA and an underlying security mechanism. As a security model, Ki-Ngā-Kōpuku, too, has a system architecture and associated security mechanism that together make the complete picture of the security model. This chapter presents the system architecture or RA of Ki-Ngā-Kōpuku as well as the contextual view of the security mechanism. First, the terms Security Model and Security Mechanism are defined in Section 5.2.1 on the next page, as it is observed that no concrete definition exists for the term Security Model or Security Mechanism, and these terms are interchangeably used in a confusing manner. The high-level conceptual view of Ki-Ngā-Kōpuku is then discussed in Section 5.2.2 on page 114, followed by the illustration of the RA that is illustrated in Section 5.3. The contextual view of the

security mechanism is also discussed in the same section. In Section 5.4 on page 122, the life-cycle of Ki-Ngā-Kōpuku is discussed to outline the deployment steps of the security model. Finally, the logical view of Ki-Ngā-Kōpuku is presented in the same section.

5.2 Distributed Security Model

The proposed security model, Ki-Ngā-Kōpuku, is distributed and decentralised. An application is split into components and distributed among Cloud servers. The components are also distributed in such a manner that no single node may contain all the components of an application, which makes Ki-Ngā-Kōpuku a decentralised system. The RA and the security mechanism for Ki-Ngā-Kōpuku work in such a way that no single point of failure exists within the architecture. The distinguishing line between a security model and a security mechanism is drawn in the following discussion, followed by the contextual presentation of the Ki-Ngā-Kōpuku architecture.

5.2.1 Security Model and Security Mechanism

Literature do not use the term ‘Cloud Security Model’ with an unambiguous meaning. The terms Security Model and Security Mechanism are not distinguished and are often used interchangeably. The terms need to be differentiated.

In CC security, two different approaches to discuss or outline security measures are found. The first approach defines the Cloud architecture and the arrangements of the elements in that architecture to provide a holistic framework for a secured CC environment. Example of such models are the multi-tenancy model of NIST, the Cloud risk accumulation model by CSA, and the Cloud Cube model proposed by the Jericho forum (Che et al., 2011). The models define the architectural arrangement and thus act

as the RA. On the other hand, the second approach discusses specific security-related mechanisms (e.g. identity management, file storage, authentication).

A security model needs to have an RA and associated security mechanism(s) to be considered as a complete security model. Thus, a security model may be defined as an amalgamation of an RA and required security mechanism(s) for the RA.

Based on the above, Ki-Ngā-Kōpuku comprises an RA and a security mechanism. A security mechanism may be architecture-dependent (where specific purpose-built architecture is a requirement for the security mechanism to function), or it may be architecture-independent (where no specific customised architecture is required for the security mechanism to function). The security mechanism of Ki-Ngā-Kōpuku is architecture-dependent, and thus the RA and the security mechanism collectively define Ki-Ngā-Kōpuku as a security model.

5.2.2 The Model

The perception of Cloud in terms of security and a security model is twofold within the realm of CC. Two different approaches are found to depict a CC security context to propose a security model or security mechanism for CC. In the first approach, the Cloud architecture is taken into consideration to draw a security model for CC, while the second approach takes specific services (e.g. authentication, file storage, identity management) into account to define a security mechanism. Thus the first approach is architecture-centric while the latter approach is service-centric. Examples of the first approach are the NIST multi-tenancy model, Cloud risk accumulation model by CSA and Cloud Cube model by Jericho forum (Che et al., 2011). These examples incorporate the elementary arrangements of a cloud architecture to define the security model. In Section 2.5 on page 64, some of the examples of both approaches towards Cloud security model or mechanism are listed.

A Cloud security model must address both the architecture-centric and service-centric concerns. Thus a complete Cloud security model should ideally be an amalgamation of an RA and a security mechanism.

As discussed above, there are two approaches under which security models and mechanisms are perceived so far within the context of CC. Ki-Ngā-Kōpuku takes a hybrid approach, which is an amalgamation of both the aforementioned approaches. Ki-Ngā-Kōpuku incorporates a security mechanism to be functional within the specific RA, thus making a complete security model for CC.

Distributed Security Model – High-Level View

The proposed model takes a hybrid approach which is an amalgamation of both approaches to security models and mechanisms discussed earlier, and incorporates a security mechanism to be functional within the specific architectural model of CC, thus making a complete security model for CC by using both the aforementioned approaches. Only addressing specific security concerns and employing countermeasures may be considered as the extent of a security mechanism; but to be considered as a security model, a solution needs to address the holistic aspects of CC both in terms of architecture and services. A Cloud security model needs to be both architecture-centric and service-centric. The proposed security mechanism is accordingly twofold: the first is to develop an RA for CC and then to employ a distributed security model within the RA to complete the picture of a security model for CC.

The proposed security model aims to eliminate a single point of attack by distributing the security mechanisms. In this way, the size and number of the targets are also distributed, making it harder to compromise. Cloud redundancy is considered a viable approach as part of proposed distributed security model.

Definition of Terms

The terms CC Architecture and Cloud Security Model are defined below:

CC Architecture: CC architecture refers to the network architecture including in-premise hardware and their underlying platforms (i.e. software tools, OS). The architecture for a CC depicts how various components are arranged and interconnected. It may also depict the interface to the outside world (for example, connectivity to the public network infrastructure).

Cloud Security Model: Cloud security model refers to the arrangement and provision within the Cloud architecture to ensure a safe computing environment. The security model may influence the architecture by implying specific provisioning and arrangement of network elements in a cloud architecture. In conjunction with the above, a security model incorporates specific security software and/or security mechanisms. A security model may be illustrated by means of a reference architecture for CC.

Contextual View of the Model

As discussed above, the proposed security model is an amalgamation of an RA and a security mechanism. The security mechanism is split into a number of components that are distributed among the cloud servers.

Assuming the following:

CA = Cloud Architecture,

C = A Cloud,

S = A Cloud Server,

M = Security Model,

λ = Component of the security model in one Cloud server,

N = Security Mechanism

The cloud architecture for any organization may be made up of one or more clouds.

Thus an organization may have C_1, C_2, \dots, C_n Clouds. Each cloud can have one or more cloud servers. So the cloud servers are S_1, S_2, \dots, S_n . As λ is the part of the security mechanism that resides in any given cloud server, all the cloud servers collectively form the total security mechanism. So in any given cloud, the distributed parts of the security mechanism could be $\lambda_1, \lambda_2, \dots, \lambda_n$. So the total part of the security mechanism for any given cloud is,

$$N = \lambda_1, \lambda_2, \dots, \lambda_n \quad (5.1)$$

As security mechanisms will reside in Cloud Servers,

$$S = \{N \in S\} \quad (5.2)$$

A cloud may consist of one or more servers. But in the specific case of Ki-Ngā-Kōpuku, there must be at least two Cloud servers. Thus,

$$C = \{S \in C | 2 \leq S \leq i\} \quad (5.3)$$

The security model may consist one or more Clouds. Thus the security model is,

$$M = \{C \in M | 2 \leq C \leq i\} \quad (5.4)$$

Thus a Cloud Architecture with the security model deployed can be expressed as,

$$\forall C [\exists M : \{(\lambda \subseteq N) \wedge (N \subseteq S)\}] \quad (5.5)$$

In order to achieve a successful Cloud breach, this would result in having multiple points for the attacker to be compromised. As the security model will distribute its mechanisms across the nodes of a cloud, and replication of individual elements provide a high level of redundancy, so if one element fails, the previously redundant component can be activated in another node. This redundant distribution eliminates the existence

of a single point of failure within a Cloud architecture.

With the establishment of a high level view of the security model, it may be important to specify the type of security the distributed security model addresses. Ahmed, Litchfield and Ahmed (2014) state that the conveyance of data for a cloud is through web services, thus the distributed security model will monitor web services of a cloud architecture along with the life cycle of the data acquired and stored through web services. For example, a worm with a lifecycle like Stuxnet or Regin (Kushner, 2013; Franceschi-Bicchierai, 2014) spreads through web services. The proposed security model is intended to repel such type of attack.

In Figure 5.1 on the next page is a conceptual illustration of the security mechanism. It is assumed that the security mechanism is split into three elements namely λ_1 , λ_2 and λ_3 . If there are three elements and three Cloud servers, each server will hold an element and redundant copies of elements also on other servers. For example, S_1 has λ_1 as its operational part and λ_2 is a redundant copy of operational element on other servers.

Referring to Figure 5.1 on the following page, as the components of the security mechanism will be distributed among the servers and all the parts of the security mechanism contribute to the security mechanism, the distribution of the security mechanism creates multiple points of attack. For an attack to be successful, all of these multiple points are required to be breached. However, the attacker will not know what elements are operational and what are merely redundant copies. Such redundant distribution eliminates a single point of attack as well as single point of failure.

In reality, a Cloud architecture may have more nodes. Besides, a new node may become live at any time. As the number of nodes grows, a mechanism for distributing the security mechanism elements is required, with the capability to decide the dominant redundant part and the dormant redundant parts of the security mechanism that would be housed in the respective server.

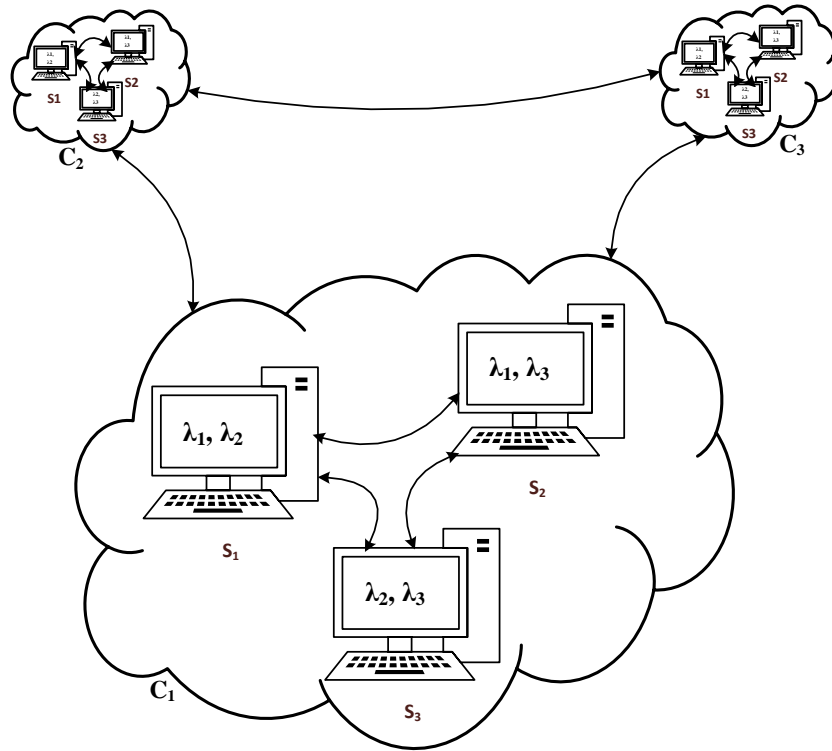


Figure 5.1: Contextual View of the Distributed Security Model

5.3 Reference Architecture and Security Mechanism

The RA for Ki-Ngā-Kōpuku is illustrated in Figure 5.2 on the next page. The RA is a two-layer architecture where the servers at the front layer interfaces the application with the end-users and the servers at the back layer holds the application components that comprise the processing core of the application. The illustration shows the front layer and the back layer of the architecture.

The architecture consists of Cloud Instance of Operating Systems (CIOSs). CIOS is the term given to the instances of OSs or platforms used within a Cloud architecture. CIOS may come in different flavours in a Cloud architecture: it may be installed on a computer/server as a stand-alone OS, or it may be installed as a VM on a type-1 or type-2 hypervisor.

An application is componentised first to make it suitable for the architecture. Any

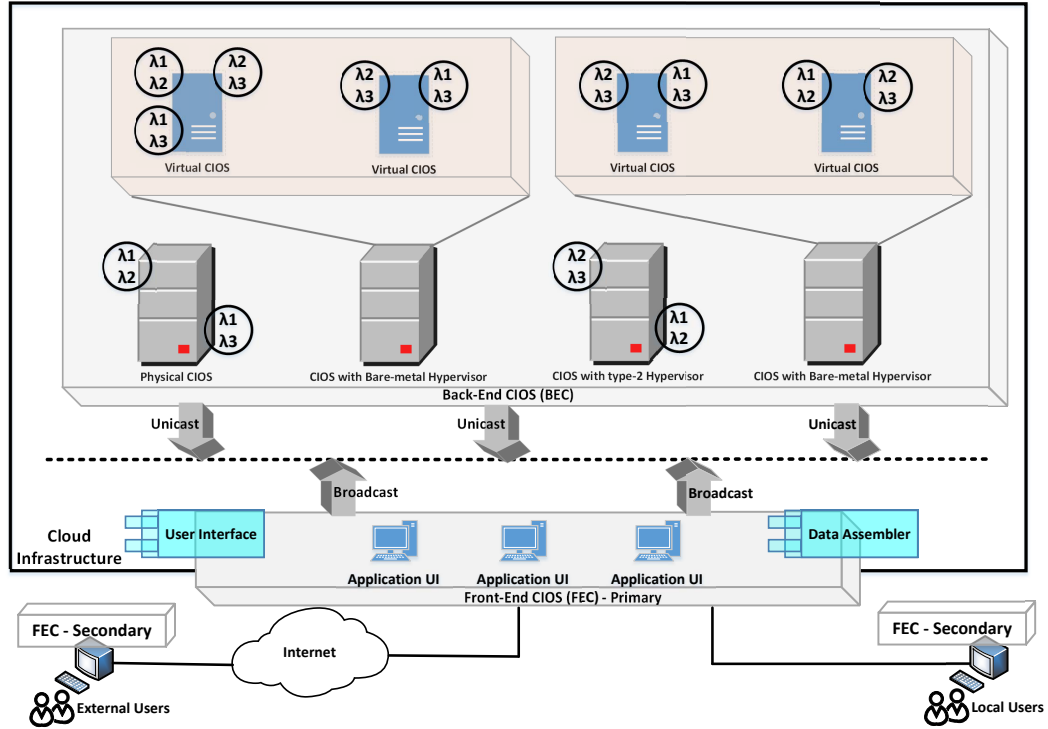


Figure 5.2: Reference Architecture for Ki-Ngā-Kōpuku

application is broadly divided into two major parts: the User Interface (UI) and the rest (processing, database and so on). The total architecture is divided into two parts: Front-end CIOs (FEC) and Back-end CIOs (BEC). Everything in an application except the UI reside in BECs which are further componentised. As illustrated, if an application is componentised to, for example, three components namely λ_1 , λ_2 and λ_3 , each of the components are housed in all the BECs which makes it distributed. The components in a CIOs are held in wrappers called nodes. Figure 5.2 suggests that a server may have various number of nodes and a number of application components may reside within each node. Referring to Figure 5.2, the circles in the BECs with application components in them are Ki-Ngā-Kōpuku nodes. No single node may contain all the components λ_1 , λ_2 and λ_3 so, to make Ki-Ngā-Kōpuku decentralised, the total application cannot reside in one single node. Thus, there will be multiple copies of any component to ensure redundancy that subsequently eliminates a single point of failure (i.e. having

the ability to take any node down will not mean having the ability to take the whole application down). The scenario at BEC is dynamic, that is, constantly changing. Any node may die and a new node may become alive at any random time. The same applies to the CIOs. This is a random feature to ensure dynamicity of the BEC scenario at any given time. Once a new CIO or a new node come alive, the component distribution takes place to the newly added node or CIO to make it part of the BEC 'family'.

FEC is divided into two parts: primary FEC and secondary FEC. The secondary FEC is the part which resides in end-user devices. Examples of secondary FEC are mobile apps and any other form of software installed in end-users' devices to connect to the Cloud architecture. Once a user initiates a request through FEC (primary or secondary), the request is broadcast to the BEC. As mentioned earlier and illustrated in Figure 5.2 on the preceding page, any specific components of an application may reside in some BECs and must not reside in all BECs. The broadcast request is picked up only by the concerned BECs who have components of the application for which the request is broadcast. The concerned BECs then process the request and verify the integrity of the processing outcome by using the security mechanism of the model. Upon integrity verification, one of the BECs is then chosen by using the underlying algorithm of the security mechanism. The chosen BEC uni-casts the processing output of the request to the FEC that originated the processing request (the requestor).

In some cases, one request from the end-user through secondary FEC may be broken down into several smaller requests at primary FEC, thus breaking down a processing request into a number of sub-processing requests. In such cases, the outcomes of each sub-processing need to be combined before delivering the outcome to the end-user. Combining the outcome of sub-processing of a larger processing is done by the Data Assembler that resides in the primary FEC.

The security mechanism of the model defines the major steps discussed above. A number of processing approaches and algorithms are required to work as part of the

security mechanism. There is a detailed discussion on the processing algorithms and processing approaches in Chapter 6 on page 125. The sequence diagram presented in Figure 5.3 portrays the contextual level of the security mechanism for Ki-Ngā-Kōpuku.

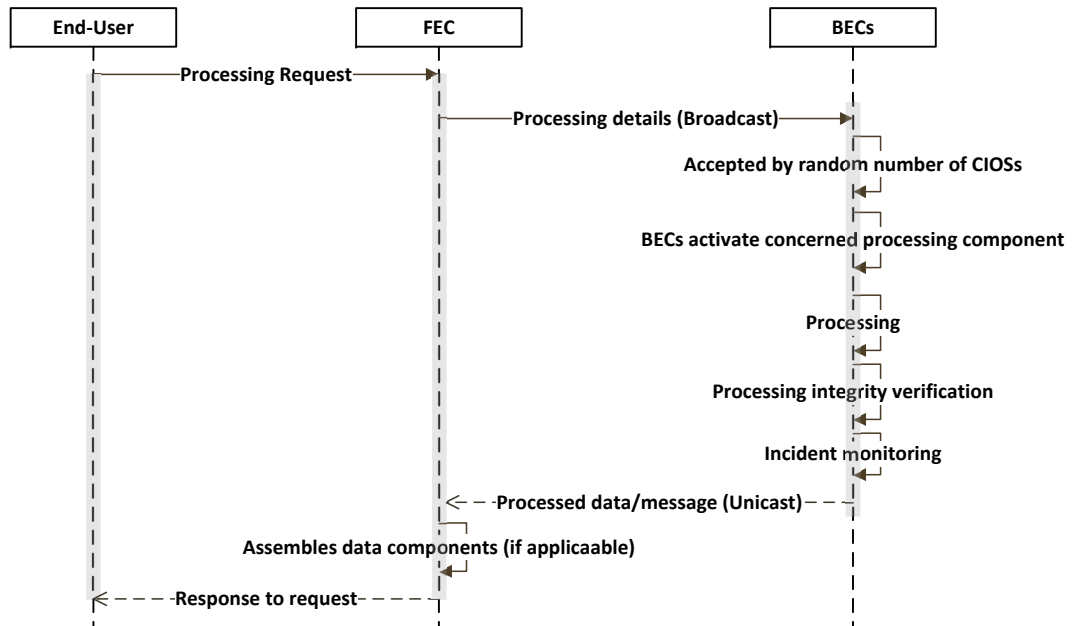


Figure 5.3: Contextual View of the Security Mechanism

5.4 Ki-Ngā-Kōpuku Life Cycle

The security mechanism for Ki-Ngā-Kōpuku makes it a pre-requisite for the applications to be componentised prior to implement/commission in a Cloud architecture adapted for Ki-Ngā-Kōpuku. Thus, componentisation is the first stage of Ki-Ngā-Kōpuku life cycle. Once the componentisation is done, the Distribution of the components among the BECs would take place, which is the next stage of the life cycle. Since one application would have a number of components yet they collectively form the total application, the collaborative interfacing among the components (hence the BECs') would be required. This Inter-component Interfacing is the third stage in the life cycle. Incident Monitoring and Response are respectively fourth and fifth (and last) stages where the former is the

mechanism to monitor the architecture for unexpected events (from security and data integrity perspective), and the latter is the response to the unexpected events. Figure 5.4 illustrates the life cycle of Ki-Ngā-Kōpuku.

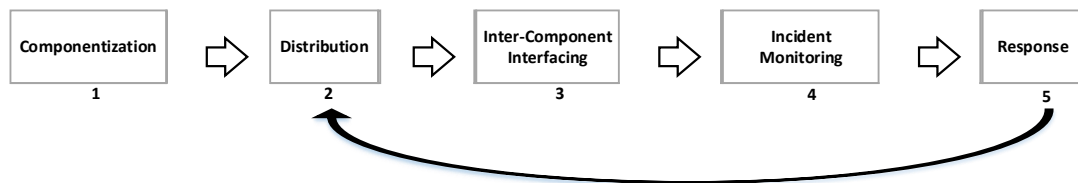


Figure 5.4: Life cycle for Ki-Ngā-Kōpuku

Stage 1 of the life cycle is a one-off effort where the rest of the stages are recurring. As the scenario at the BEC premise is dynamic and constantly changing, introduction of a new BEC would trigger the stages from distribution to the subsequent stages of the life cycle.

The communication diagram depicted in Figure 5.5 demonstrates the logical view of the distributed security model.

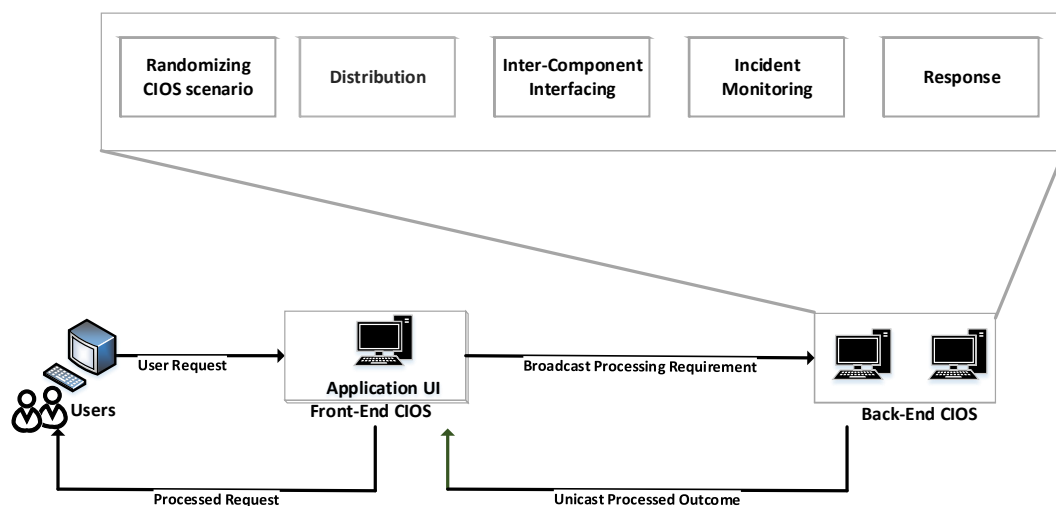


Figure 5.5: Logical View of Ki-Ngā-Kōpuku

The functions of the BECs are somewhat similar to the stages of the life cycle except

componentisation which is done prior to implement an application within Ki-Ngā-Kōpuku architecture. Thus, componentisation is not part of a process in the operational scenario of Ki-Ngā-Kōpuku, rather, it is a pre-requisite. randomising the scenario at BECs is one of the functions which is not to be perceived as a stage of the life cycle, as this function is triggered in a random manner only when a new CIOS becomes alive within the BEC premise.

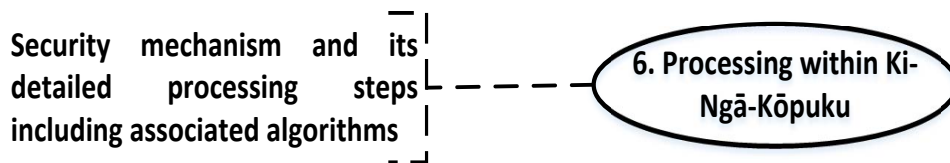
5.5 Conclusion

Ki-Ngā-Kōpuku is not only distributed, but also decentralised. It is found that few security mechanisms incorporate limited aspects of distributed characteristics, but they are either not termed distributed or the distributed nature of security is not given explicit attention. As well, a security model or mechanism that is purely distributed and decentralised in nature to qualify to be termed a “Distributed Security Model” is not found. Thus it is believed that the proposal of a distributed security model for CC is an innovative approach, and to the best of our knowledge, is the first of its kind.

The previously discussed SRS is the baseline for the detailed specifications of Ki-Ngā-Kōpuku. The discussed architecture details the architecture-specific requirements for Ki-Ngā-Kōpuku. As the security model consists of the discussed architecture as well as the underlying security mechanism, the complete picture of Ki-Ngā-Kōpuku would require explanation of the security mechanism or the processing approaches within Ki-Ngā-Kōpuku. In this chapter, the security mechanism for Ki-Ngā-Kōpuku is discussed at contextual level. The processing steps or the security mechanism for Ki-Ngā-Kōpuku is discussed in the next chapter.

Chapter 6

Processing within Ki-Ngā-Kōpuku



6.1 Introduction

This chapter explains the security mechanism of the distributed and decentralised security model. In this chapter, processing approaches and associated algorithms for Ki-Ngā-Kōpuku are discussed. All the discussed processing steps and algorithms collectively form the security mechanism for Ki-Ngā-Kōpuku. The process of componentisation and component distribution is discussed first, followed by the steps and algorithm for inter-component interfacing (how the components of an application may communicate among themselves). The aspects of incident monitoring (i.e. incident detection and response) are discussed afterwards. Section 6.2.1 on page 127 addresses the componentisation of the security mechanism, whereas Section 6.2.2 on page 128 discussed the elements of a node. The process of adding a new node is discussed

in Section 6.2.3 on page 131. The steps of inter component interfacing is addressed in Section 6.2.4 on page 136, and Section 6.2.5 on page 142 is about how the process of incident monitoring and response to incidents are embedded into the security mechanism.

6.1.1 Assumptions

The research presented is the first phase development of the blueprint of Ki-Ngā-Kōpuku.

A number of assumptions are made at this stage:

- Componentisation is done prior to deployment. This is not automated.
- At least $(N + 1)$ number of nodes are deployed with distributed components, where N is total number of component, and $N \Rightarrow 3$. That is, if there are three components λ_1 , λ_2 and λ_3 , at least 4 nodes are deployed as the initial scenario where one node contains λ_1 and λ_2 ; and the other node contains λ_1 and λ_3 (or λ_2 and λ_3), and so on. This initial scenario is assumed for Ki-Ngā-Kōpuku to start functioning.
- Adding a new node triggers component distribution (described in Section 6.2.3 on page 131). The addition of new node is assumed to be a manual process at this phase.
- In the Cloud architecture, only one application is deployed using Ki-Ngā-Kōpuku. While this is not the case in real life scenario and a cloud server may host a number of different application depending on the service deployment model (i.e. SaaS or PaaS), assuming so helps to simply the illustration of the principles of Ki-Ngā-Kōpuku and its associated processing approaches and other features. The previously illustrated Ki-Ngā-Kōpuku architecture is based on this assumption, where the components of one and only one application is shown as residing in different BECs.

6.2 Security Mechanism

Ki-Ngā-Kōpuku requires a number of processes and algorithms. These processes and algorithms collectively form the security mechanism. The major functions of the security mechanism are component distribution (when a new CIOs is added to the architecture), inter-component interfacing (communication and collaboration among the nodes and components) and incident monitoring (monitoring and response). The following discussions address Ki-Ngā-Kōpuku processes.

6.2.1 Componentisation

The componentisation of an application is dependent and specific to that application. The componentisation of an application is decided based on the functionality of the application to be componentised. While Ki-Ngā-Kōpuku defines the generic steps for componentisation. For any application, it is important to decide total functionality of the respective application as well as the data requirements. Apart from these, total number of requests that could be initiated by the end-users needs to be listed. Thus, the following two components should be decided before componentisation of an application may take place:

Functionality: An application may be developed using any tool or programming language. Ki-Ngā-Kōpuku nodes will act as wrapper for the components of that application once divided into several parts. To decide, how an application is to be divided, it is crucial to study the functionality and the modularity of the application.

Interface: The functions or modules of the components that would interface outside the scope of the application needs to be identified. It could be a user interface or an interface to other system.

6.2.2 Elements of a Node

As mentioned earlier, a CIOS is thought to have only one application (to be precise, its components) deployed in it. However, in reality, a CIOS may house different applications. If an application is deployed using Ki-Ngā-Kōpuku, the components of that application resides in Ki-Ngā-Kōpuku nodes.

Node: A Ki-Ngā-Kōpuku node is the logical wrapper that contains the components of the application deployed. A CIOS may contain any number of nodes, and all the nodes do not necessarily contain identical components. Ki-Ngā-Kōpuku nodes have same structure but the application components held within the nodes are random.

The above definition implies that all the nodes have same construct. A node differs from another node in such that they do not necessarily contain exactly same components. For example, if an application is divided into three components λ_1 , λ_2 and λ_3 , and while a node may contain λ_1 and λ_2 ; any other node may contain either λ_1 and λ_2 , or λ_1 and λ_3 , or λ_2 and λ_3 . Figure 6.1 illustrates the elements of a node.

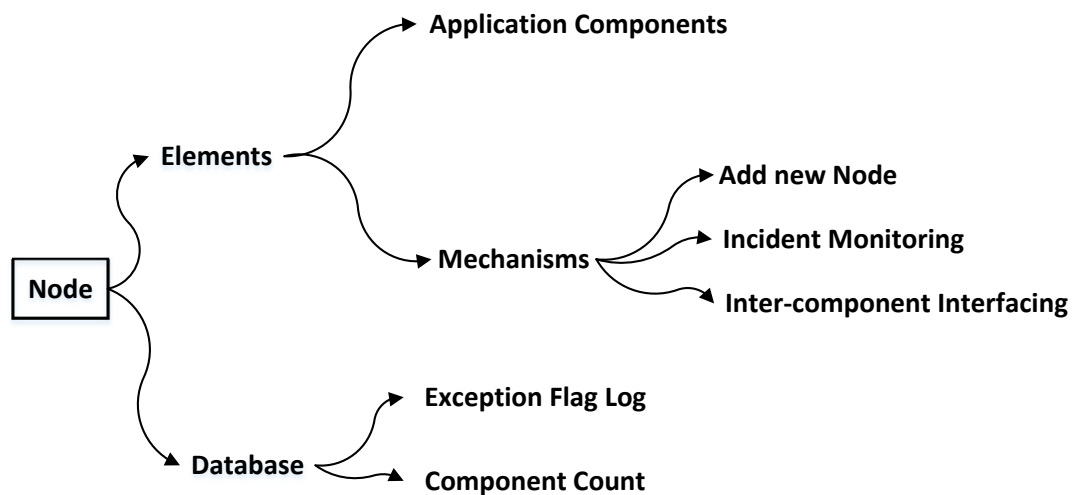


Figure 6.1: Elements of a Node

A node consists of all the components and modules to carry out all the functionalities of Ki-Ngā-Kōpuku. As illustrated in Figure 6.1, all the nodes have equal capability

and no node supersedes other nodes. A node consists of application components, databases and mechanisms. A node contains application components – these are the components of the application deployed using Ki-Ngā-Kōpuku. In addition, a node contains mechanisms for adding a new node as well as mechanisms for inter-component interfacing and incident monitoring. These processes are discussed later in this chapter. Apart from the above, a node contains the following two databases:

Component Count: This is the list of the total number of components distributed. The nodes will keep track of the total number of existing components distributed so far. This is to ensure that the component distribution is done in a nearly balanced manner so that no single component outweighs in number compared to other components. For example, if an application has three components, say λ_1 , λ_2 and λ_3 ; and if the number of these components at any given time is respectively 35, 34 and 37, then the distribution of components can be considered as nearly balanced. However, if the existing components number were, let's say 30, 24 and 76, then it would be an imbalanced distribution. Keeping a track of total number of existing component would help a nearly balanced distribution. The mechanism of 'component count' is discussed in 'component selection' process later in Section 6.2.3 on page 131. This file also keeps two additional parameters: total number of components and total number of components to be distributed where the latter is always less than the former. If a node goes down, the component list is not required to be updated by decrementing the number of components distributed so far. The validation and illustration of this is addressed in Chapter 7.

Exception Flag Log: This is the log file that records all exceptions. This is illustrated in rest of the discussion in this section, where various processing approaches are explained.

A node initially contains all the parts illustrated except the application components.

When a node want to have application components, it sends request to other nodes so that other existing nodes distribute the components. The process a adding a new node is discussed below. The initial Ki-Ngā-Kōpuku node (with all default functionality but the application component) may be called as the default node. A default node is a standard wrapper and the building blocks of Ki-Ngā-Kōpuku. A default node is ready to receive any application component, to work as an application node for the application deployed, and in collaboration with other application nodes. To sum up, a default node is a node that is a standard wrapper node ready to use for application deployment. A default node transforms into an application node when it receives components of an application from other application nodes containing components of the same application. As described later in Section 6.2.3 on the next page, it is common within Ki-Ngā-Kōpuku context to introduce new default nodes that request components from application nodes to transform itself into an application node. This is illustrated in Figure 6.2.

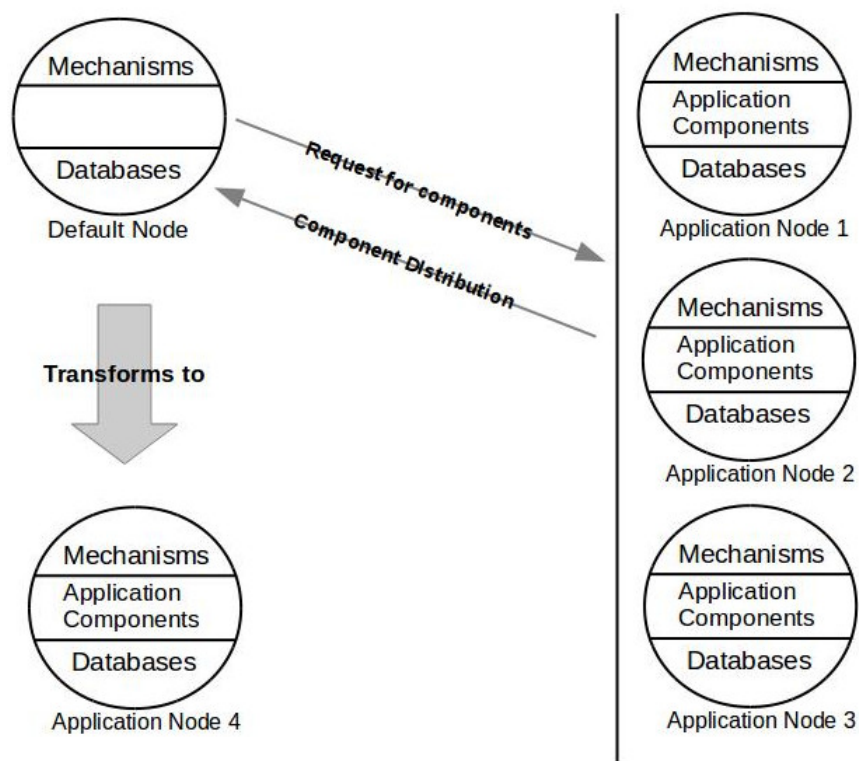


Figure 6.2: Transformation of a Default Node into Application Node

As Figure 6.2 on the previous page illustrates, upon successful components distribution, a default node becomes a group member of the application nodes. The number of application nodes is insignificant here (i.e. number is not relevant), and for illustration purpose only. The number of existing application nodes are random provided the base constraint is not violated (the minimum number of nodes must be greater than N , where N is the total number of application components).

The following formally defines Default Node and Application Node:

Default Node: A Ki-Ngā-Kōpuku node that has all elements of a node except application component(s). A default node is ready to take any application components to turn itself into an application node. Databases in a default node are empty with no information.

Application Node: When a default node accepts application components from other nodes, it becomes part of the family of the nodes that collectively deploy the application. Upon request, a default node may receive random components from other nodes. An application node may contain components for only one application.

6.2.3 Add New Node

At BEC level of the architecture, the scenario is random where a new CIOS may become active (commences the process) at any time. When a new CIOS comes alive, the existing BECs need to make it part of the BEC family. This is done by distributing components from the existing BECs to the newly added server. It is important to note that, when a new CIOS comes alive, it has pre-installed Ki-Ngā-Kōpuku default nodes. When a CIOS becomes alive, the node(s) within it sends request to other CIOSs to distribute application component so that they can become application node from default node. A

default node may reside in an existing CIOS or in a new CIOS. The flowchart of the steps to add a new CIOS (hence at least one new node) is illustrated in Figure 6.3.

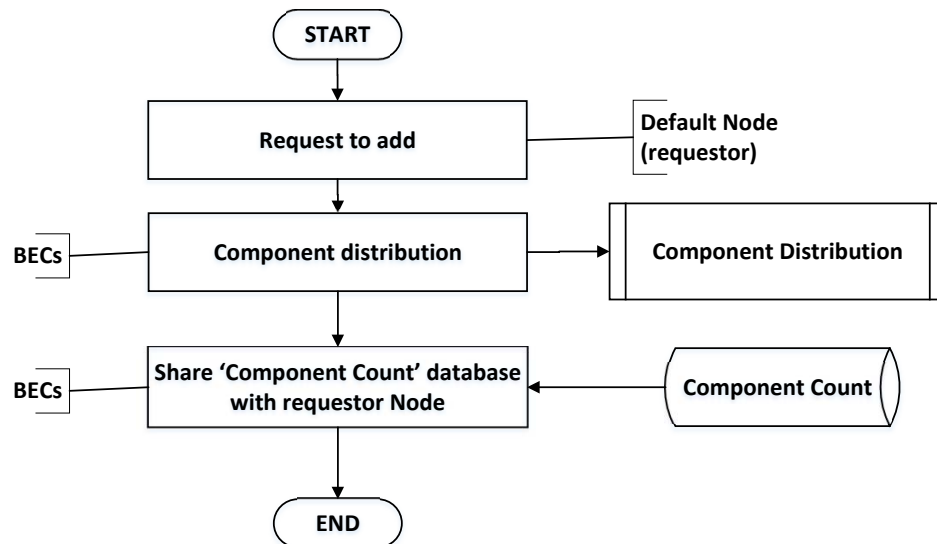


Figure 6.3: Add New Node

A default node acts as the requestor to become an application node. Upon request, existing application nodes (from the same CIOS or from different CIOSs) initiate a sub process “Component Distribution” as illustrated in Figure 6.4 on the next page. Once component distribution is successfully accomplished, the application nodes share component count database with the default node which marks the end the of the process of adding a new node and subsequently the default node becomes an application node. The process of component distribution is discussed below.

The first step in component distribution is component selection. The steps in component selection is discussed below and illustrated in Figure 6.5 on page 135. To carry on with rest of the discussion on component distribution, it may be enough to note here that, the steps in component selection helps to determine which components are to be distributed to a newly added node. This is to ensure that same components are not being distributed repeatedly while some other components are never distributed. This is also to ensure that any given node do not contain all the components of an application.

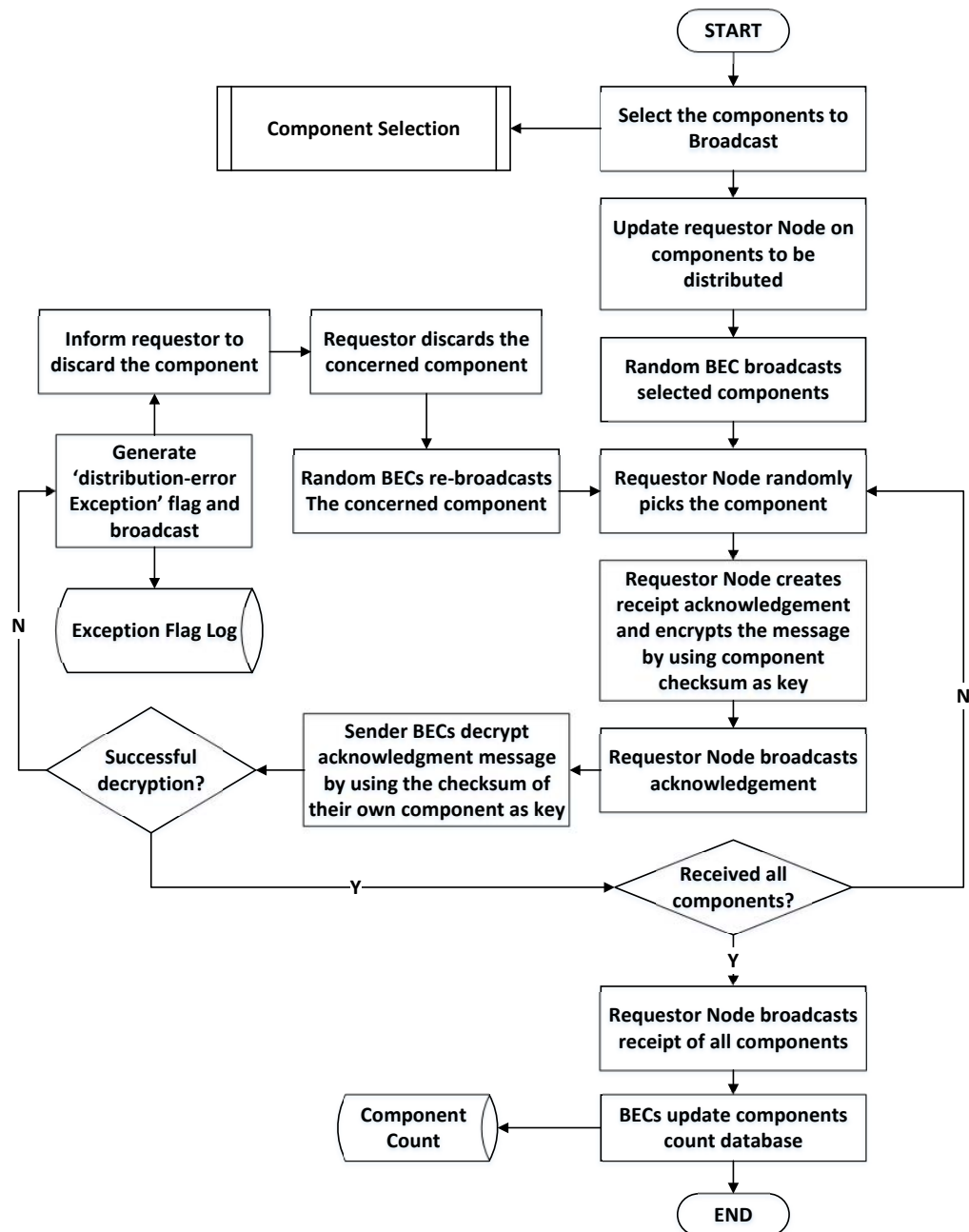


Figure 6.4: Component Distribution

However, upon component selection, the application nodes know which components to distribute to the requestor default node. Application nodes then update the requestor default node on the components that is to be distributed to it and requestor default node

now knows which components it is to receive to become an application node.

After that, random BECs (or the nodes contained in the BECs) broadcast the components, and the requestor default node randomly picks only one copy of each of the component it is informed to receive. The requestor default node then creates checksum or hash value of each component received, and uses this as a key to encrypt a standard acknowledgement message. The requestor default node then broadcasts the encrypted message to application nodes without sharing the key. The application nodes are able to decrypt the acknowledgement message by using the checksum or hash value generated from their own contained component. Since the components are same, the generated checksum or hash value for any component must match for any node having that component. If the decryption is successful, it indicates successful transfer of a component from application node to requestor default node. Other, a failed decryption would mean integrity issue of the component received by the requestor default node, and an exception flag is generated and logged in the exception flag log database. In the case of an exception, the process of re-transferring the same component described above is repeated.

Upon successful transfer of all the components, the existing nodes update their "Component Count" database by increasing the value of the components by one only for the components that are distributed (as discussed earlier, all the components of an application will not be distributed in one distribution based on the constraint that a node may not contain all the components of an application). When a default node becomes an application node, it contains only one copy of the components. For example, if an application is divided into three components λ_1 , λ_2 and λ_3 , then a node may contain any two of the components and will not have duplicates of the components it contains (e.g. having two copies of component λ_1 , and so on).

Once the components are successfully transferred to the requestor default node, and the application nodes have updated their components count database, the process

of adding a node is marked successfully done by sharing the updated component count database with the requestor default node. Now the default node has become the application node and it is not a default node any more. Figure 6.5 illustrates the process of component count.

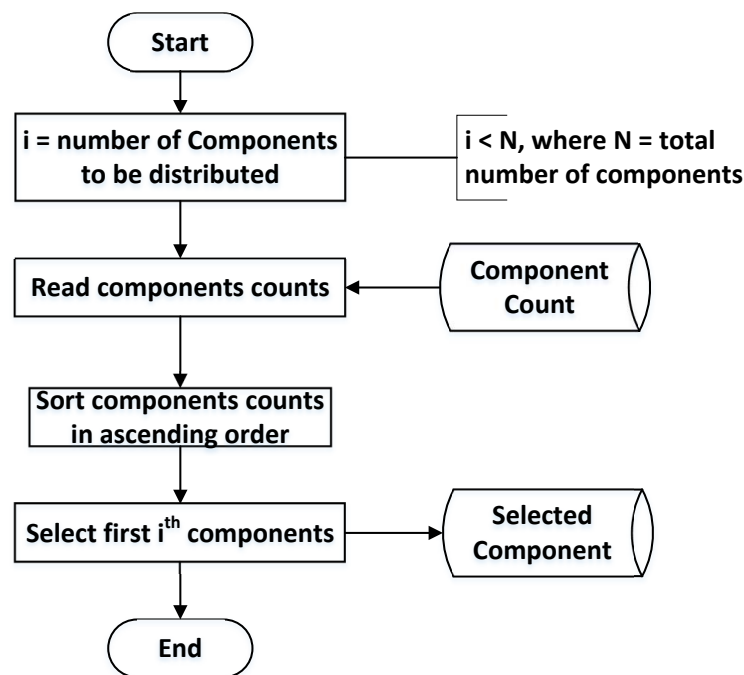


Figure 6.5: Component Selection

Total number of components for an application is fixed by the developers during the componentisation phase, so is the total number of components to be distributed in one distribution while adding a new node. For example, if an application has five components, the developers may decide to distribute only three components at any distribution. This will prevent one node from having all the components of the application. At the same time, it is imperative to avoid distributing the same component repeatedly. This is done and ensured by the Component Selection process. If i number of components are to be selected, the component selection process will read the component count file and then sort the component count in ascending order. Then the process will

select the components associated with the first i number of counts from the sorted list.

The above is illustrated below using an example. It is assumed that there are three components named λ_1 , λ_2 and λ_3 ; and the component count is $\lambda_1=12$, $\lambda_2=10$ and $\lambda_3=10$. It is also assumed that the developers have decided to distribute two components for every distribution. Thus, here $i = 2$. The Component Selection process will sort the component count and the list will be $\lambda_2=10$, $\lambda_3=10$ and $\lambda_1=12$. Since $i = 2$, the first two components from the sorted list (that is, λ_2 and λ_3) will be selected to be distributed to the newly added node. This will ensure all the components are being distributed evenly and no single component outweighs others in number at any given scenario within the BEC premise.

6.2.4 Inter Component Interfacing

Ki-Ngā-Kōpuku application nodes contains components of an application. All the components scattered in different nodes collectively work in such a manner to give an impression as if the application is an integrated stand-alone one, and thus keeping the aspects of componentisation transparent to the end-user. The end-users do not feel any difference when using an application deployed using Ki-Ngā-Kōpuku compared to an application that is not componentised and not deployed using Ki-Ngā-Kōpuku. An application is consisted of the application nodes. As a result, there needs to be interfacing and communication among the application nodes and its underlying application functions, in order for the application to be able to function as it would if it were deployed (installed) in traditional way and without making it decentralised and distributed using an architecture like Ki-Ngā-Kōpuku. In this section, the interfacing among the components is discussed.

As example, it is assumed that an application is divided into four components λ_1 , λ_2 , λ_3 and λ_4 . The components are randomly distributed to a random number of nodes.

If the number of components are N , then the number of nodes must be greater than N . For example, if an application is made up of four components, there must be at least five nodes existent in the architecture. This is the lower limit of the nodes, and there is no theoretical upper limit. Also, if X is the number of components, the highest number of components a node may have is $(X - Q)$ where $1 \leq Q < X$.

The smaller the value of Q in the above condition, the better randomness can be achieved. The components are distributed in such a manner that no single node may contain all the components, and there must be redundant copies of each component. Some of the components are active and some of the components are non-active but awake. An illustration of nodes and components are illustrated in Figure 6.6 on the following page, where the active components are shown in dark colour and the non-active components are shown in light colour. Thus, node 1, 2, 3 and 4 are active and the rest are non-active in the illustration.

Taking down one node will not make all the components unavailable, because the redundant copies are existent in other nodes. This ensures that there is no single point of failure. When a processing is associated with a component, for example λ_1 , an active node that has λ_1 will carry out the processing. If an active node with λ_1 cannot be found when required, a non-active node with that component will become active. On top of having no single point of failure, this exhibits the self-healing feature of Ki-Ngā-Kōpuku.

Figure 6.6 on the next page presents a conceptual example of how components may work within Ki-Ngā-Kōpuku. It also shows a possible approach to divide an application into several components (componentisation). It should be noted that the illustration in Figure 6.6 on the following page shows that all the nodes have only one component though it does not have to be the case, as discussed earlier.

Figure 6.6 on the next page shows an application componentised based on its functions. It is assumed that the application has a total of six functions Function-A to

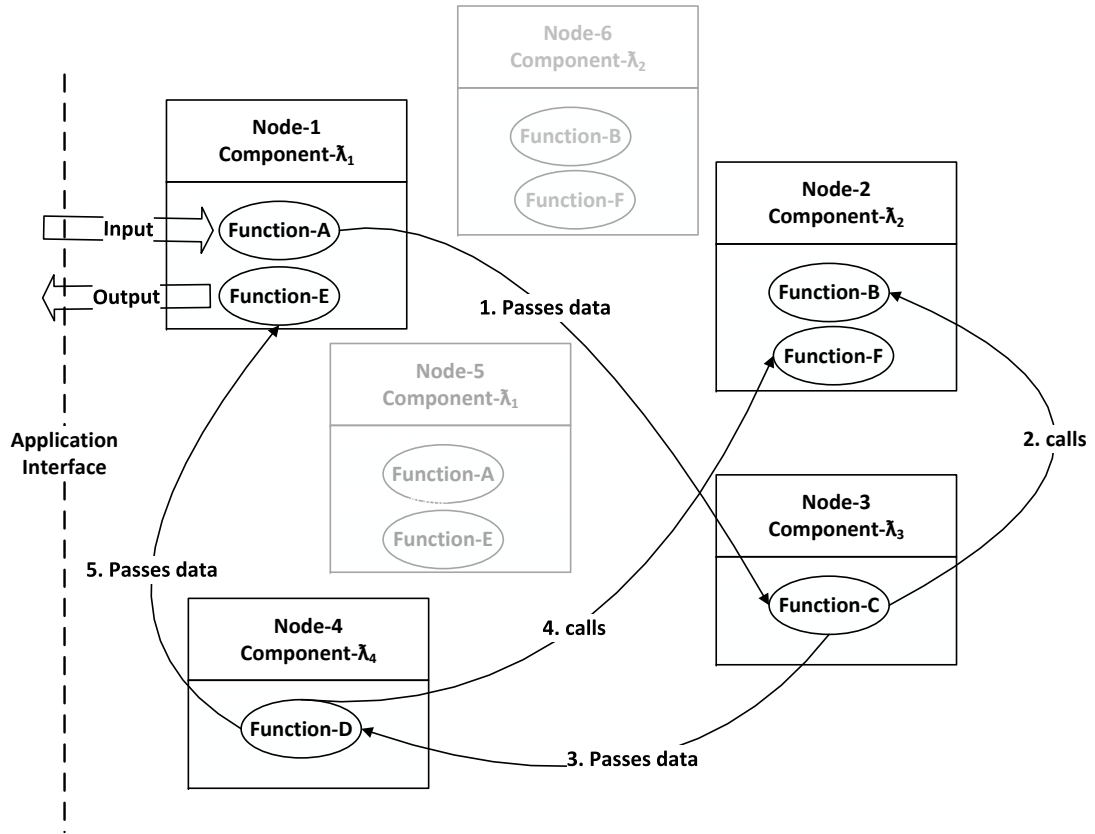


Figure 6.6: Function Calls among Components

Function-F as illustrated. The application is divided into four components λ_1 , λ_2 , λ_3 , λ_4 . A component may contain one or more functions of the application. It is assumed that each node contains one component (but this is not a constraints, as a node may contain more than one component as mentioned earlier). There must be redundant copies of all the components. For example, Node-5 is a replica of Node-1 that contains the component λ_1 and is non-active. If Node-1 becomes unavailable for any reason, Node-5 will become active as a substitute of Node-1 and there will be force creation of further replica of the node. Figure 6.6 is for illustration purpose only and does not represent a complete scenario (for example, as it does not show redundancy for all the nodes or components). However, careful examination of the illustration would reveal that the randomly scattered components and functions collectively make the whole

application, and the active components works collaboratively to behave as a single application. To do so, some components may take user input and throw output to the user. Function in a node either passes message to another function residing in another component which in turn resides in another node. A function may also call another function to carry out some sub-process. Thus data sharing and message passing among the functions in different components as in Figure 6.6 on the previous page show how an application can be componentised based on its functions.

For an application to function that is deployed in a decentralised and distributed manner using Ki-Ngā-Kōpuku, the components need interfacing, collaboration and communication capability among themselves. All the nodes and the components contained in the nodes collectively define the scope of an application, though in a redundant manner with no single core point of failure. This makes it imperative that there needs to be some procedures or mechanism for the components to be able to communication among themselves. When a component is in communication with another component, it may reside in the same node. Since the distribution of the components is random, there is no way to predict which node would contain which components. As a result, when a components is required to communicate with other components (essentially it is by means of function calls among the components, as depicted in Figure 6.6 on the preceding page), the sought components may reside in the same node or in another node. Thus, communication and coordination among the components are essentially achieved via inter-component interfacing, which may be inter-node interfacing as well.

The process of inter-component interfacing is illustrated in Figure 6.7 on the next page.

Inter-component interfacing involves a sub-process called target coupling (illustrated in Figure 6.8 on page 141). By target coupling, two components aim to establish a session to start communication. However, if target coupling is unsuccessful, an

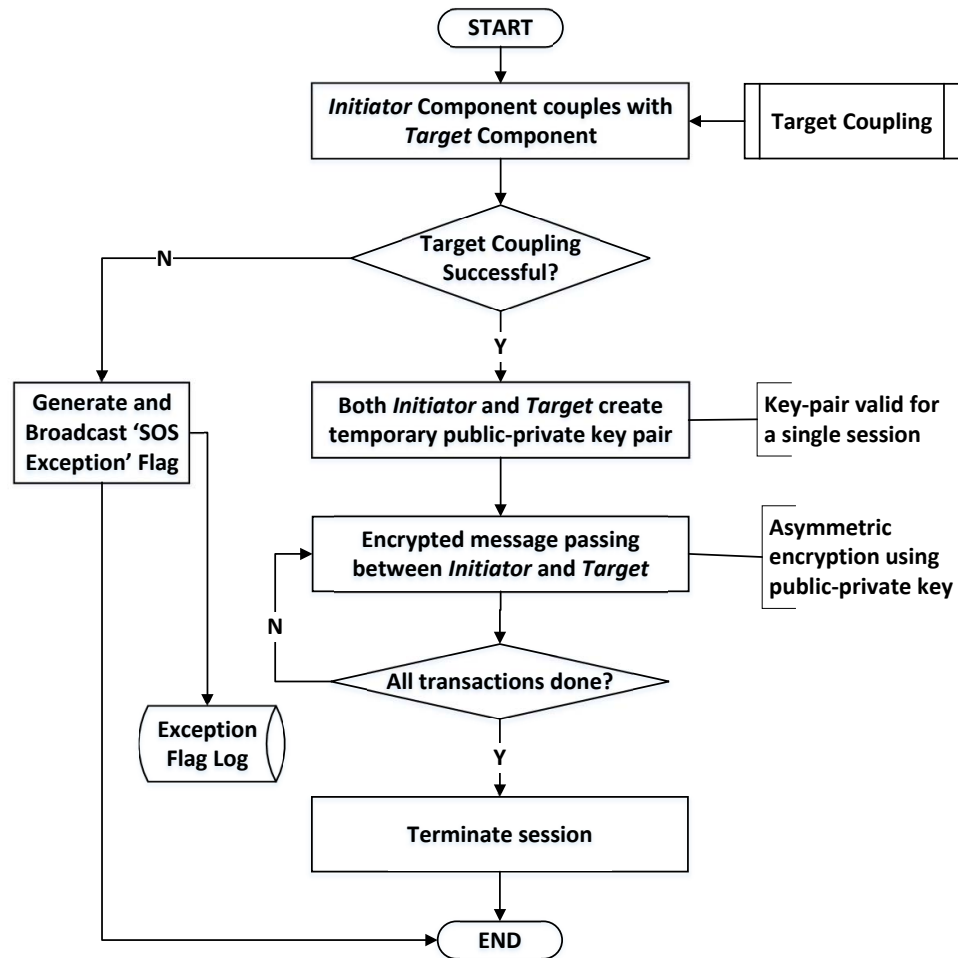


Figure 6.7: Inter-Component Interfacing

exception is generated and logged into the exception database (exception flag log). On the other hand, successful accomplishment of target coupling triggers actual intended communication between the components. The requestor node or components can be named as the Initiator, and the requested node or component as the Target. Both Initiator and Target create their own temporary public-private key pair to carry on with the session by using asymmetric encryption. The key-pair is valid for one session only and once the session is closed, the key-pair for both the Initiator and Target is expired. For every new session, a new key pair is required. Any node or component may act as Initiator or as target depending on whether the node or component is a requestor or

requested for any specific session. Once all transactions are done, both the Initiator and Target terminates the session.

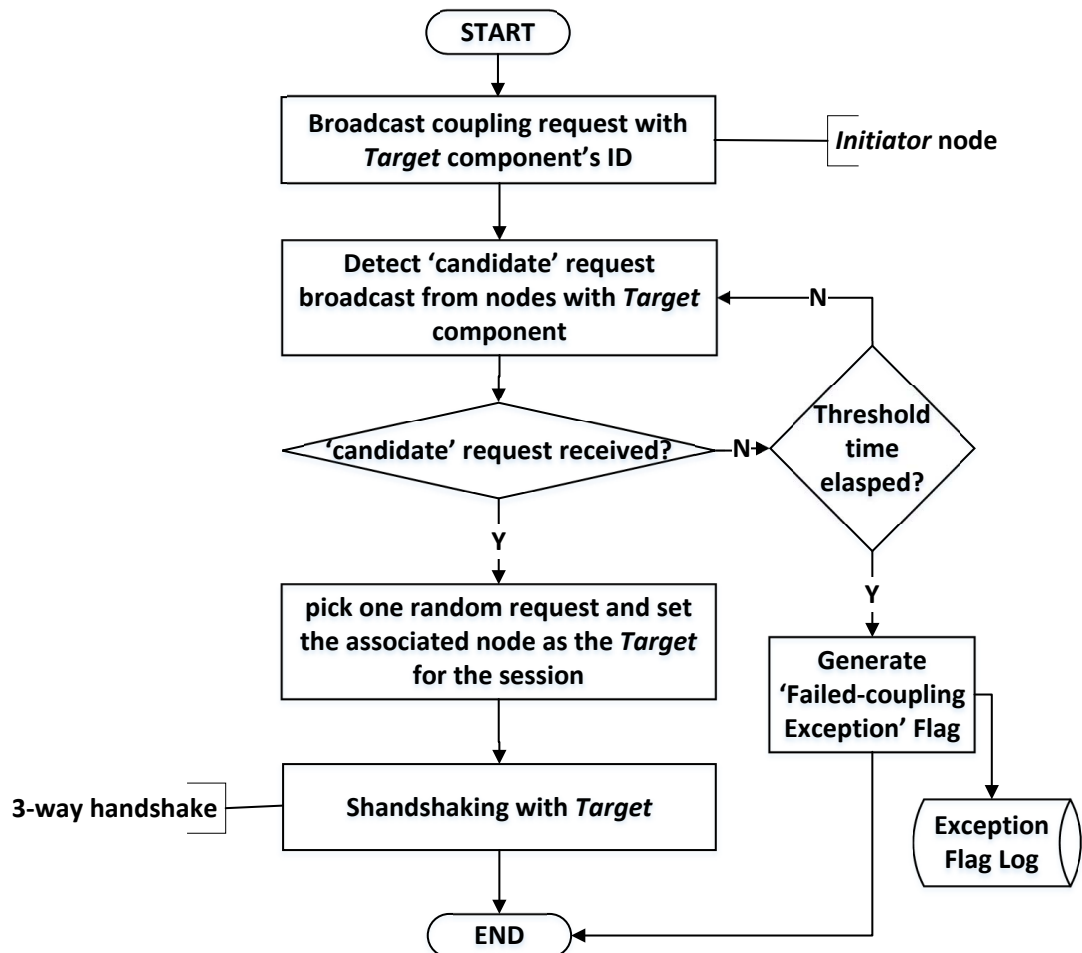


Figure 6.8: Target Coupling

Figure 6.8 illustrates the process of target coupling which is a pre-requisite for successful and consistent inter-component interfacing. If target coupling is unsuccessful, inter-component interfacing and subsequent communication between two components or nodes may not take place. As Figure 6.8 depicts, the Initiator first broadcasts for the components it is looking for to couple with. The component the Initiator is looking for may ideally be found in a number of other nodes, which would result in a number of other nodes that contain the requested component to respond to the request broadcast

by the Initiator. This is the candidate request from the nodes that wants to be the Target for the requested session. The Initiator wait for a pre-specified threshold time and if no candidate request as response is received from other nodes, the Initiator node generates an exception flag, as this would be an exception scenario and may be an indicator of inconsistency or other malicious activity within the Ki-Ngā-Kōpuku architecture. In a normal scenario, the Initiator would receive a number of responses from which it will randomly pick one of the candidate request and will set the originating node of that request as the Target for the session. This is followed by a three-way handshake and successful completion of the handshaking would mark completion of the process of Target Coupling. The rest of the steps of the process of inter-component interfacing as illustrated in Figure 6.8 on the previous page progresses from this point onwards.

6.2.5 Incident Monitoring and Response

Incident monitoring is not an isolated process for Ki-Ngā-Kōpuku. As illustrated in the security mechanism and the processes throughout the chapter, any exception or unexpected event is monitored in real time. The Ki-Ngā-Kōpuku nodes come with the capability of such monitoring and any exception is logged into the exception file. The exception file can then be analysed and used to determine the type of action required for specific exception flag/message generated.

However, the actions based on the exception flags would be more like cleansing rather than fixing Ki-Ngā-Kōpuku nodes or the architecture. This is ensured by the self-healing feature of Ki-Ngā-Kōpuku through redundancy and having no single point of failure. If a node or components is compromised or goes down, other nodes simply exclude or ignore that node, since excluding any component or node would not mean excluding any part of the system due to redundancy.

A response based on an exception flag or message generated may be taken. The

development of the response mechanism is considered as a future enhancement and discussed further in Chapter 9 on page 185 under Section 9.2 on page 188.

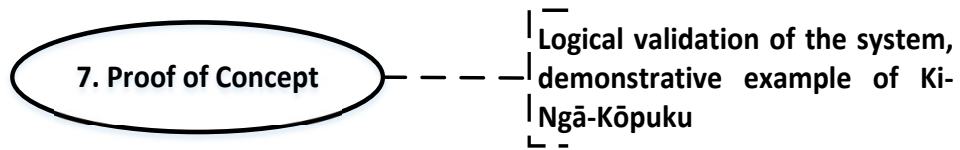
6.3 Conclusion

This chapter defines the security mechanism for Ki-Ngā-Kōpuku. The security mechanism and Ki-Ngā-Kōpuku as a whole security model complements any other security model or mechanism and may accommodate other security mechanisms. For example, for Target Coupling Ki-Ngā-Kōpuku security mechanism may use any existing approach to three-way handshaking algorithm; or for inter component interfacing, it may use any existing asymmetric encryption algorithm. This enhances the interoperability of Ki-Ngā-Kōpuku as a security model.

In the next chapter, the Proof of Concept of the model is presented, where validation of the conceptual model and other aspects of the security mechanism are addressed in the discussion.

Chapter 7

Proof of Concept



7.1 Introduction

The Proof of Concept of Ki-Ngā-Kōpuku is demonstrated in this chapter through validations of the concept and the model. The concept of the security model is presented by means of Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA) to show both the conceptual construct and the validation of the concept of Ki-Ngā-Kōpuku. Further validation of the model is done by developing a Turing machine equivalent for Ki-Ngā-Kōpuku that shows that the concept of the security model is Turing complete (Ahmed et al., 2016).

Ki-Ngā-Kōpuku security mechanism consists of a number of processing approaches and algorithms that collectively define the security mechanism for the security model. The proof of concept on the concept of the security mechanism is done by logical

demonstration of an example application. The illustration of the example application shows how it can be deployed using Ki-Ngā-Kōpuku, and then further validation is shown through logical constructs.

The validation also includes the illustration of components distribution that shows a logical simulation of progression of component distribution. This demonstration shows how the system gradually grows to be decentralised and distributed with redundancy resulting in eliminating single point of failure.

Section 7.2.1 on the following page shows the validation using automata. The Turing Machine is presented in Section 7.2.2 on page 149. Validation through logical demonstration of example application is presented in Section 7.2.3 on page 151, and the logical simulation of distribution is described in Section 7.2.4 on page 154. Section 7.3 on page 160 discusses further validation of the security model in addition to the validation presented in earlier sections. A number of recent Cloud and other data breaches are taken into account to analyse whether Ki-Ngā-Kōpuku can provide any protection against them.

7.2 Validation

Model validation for the system is done using formal methods. The NFA and its equivalent DFA on the concept of the security model is discussed in this section. Model validation as well as the validation for the security mechanism establish the feasibility for Ki-Ngā-Kōpuku to be deployed in real-life scenarios.

In addition to model and security mechanism validations through NFA, DFA and logical simulation, it is also shown that the proposed model is Turing complete and subsequently satisfies Church-Turing theorem. The theorem states that the Turing completeness of a concept (or algorithm) ensures the computability of the concept (or algorithm) (Yao, 2013). The Turing completeness thus further validates Ki-Ngā-Kōpuku.

These validations are discussed below.

7.2.1 Model Validation

In this section, the concept of the security model is presented by means of Non-deterministic Finite State Automaton (NFA) and its equivalent Deterministic Finite State Automaton (DFA).

In reality, the birth and functioning of a cloud may be initiated with one server and more servers can subsequently be added. The security mechanism is distributed if there are at least two components within any given server. Thus for a server, the security mechanism deals with three possible inputs: no change, addition of a component and deletion of an existing component.

Figure 7.1 (Ahmed et al., 2016) is the NFA presentation for the security mechanism discussed.

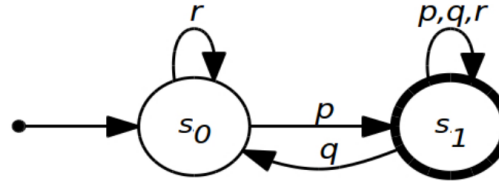


Figure 7.1: NFA Representation of the Distributed Security Model

The NFA has two states s_0 , s_1 and three input functions r , p and q . The states and inputs are:

r = No change: an input function which has no effect on the current state of the security mechanism.

p = Add server: an input function for the addition of a component.

q = Delete server: an input function for the deletion of a component.

s_0 = a state where there is only one server in the security mechanism, (λ_1, λ_2) .

s_1 = the acceptance state, $(\lambda_1 + \lambda_2)$.

Two servers in the security mechanism, which makes it distributed. More servers can be added.

The NFA can be represented as $M = (Q, Z, R, q, F)$ where:

M = the NFA

Q = states of the NFA $\{s_0, s_1\}$

Z = input symbols $\{r, p, q\}$

F = acceptance state $\{s_1\}$

q = initial state $\{s_0\}$

R = transition $R : (Q * Z) \rightarrow Q$ (see Table 7.1 on page 150)

$\#$ = a state of $\{s_0', s_1', s_0', s_1'\}$

This is a failure and from this state there is no return.

$$\begin{pmatrix} R & p & q & r \\ s_0 & s_1 & NULL & s_0 \\ s_1 & s_1 & s_0, s_1 & s_1 \end{pmatrix} \quad (7.1)$$

Matrix 7.1 is a 3×4 matrix and represents the transition function R of the NFA, where:

Elements in column 0 represent the initial state of NFA $\{s_0, s_1\}$

Elements in Row 0 represent the input symbols $\{p, q, r\}$

The remainder of the elements represent the final state. For example, if s_0 receives an input p , then it moves to s_1 , represented as $R : (s_0 * p) \rightarrow (s_1)$. Similarly, if s_1 receives an input q , then it moves to (s_0, s_1) , represented as $R : (s_1 * q) \rightarrow (s_0, s_1)$.

Figure 7.2 on the next page is the equivalent DFA of the NFA presented above.

The number of states in the DFA are the power set of the number of states in the NFA.

If the total number of states in an NFA are n then the number of states in a DFA would be $2^{**}n$. For the above DFA the total number of states are 2^{**} (number of states

in the NFA $\{s_0, s_1\}$. Therefore, the number of states in the DFA are $2 \times 2 = 4$.

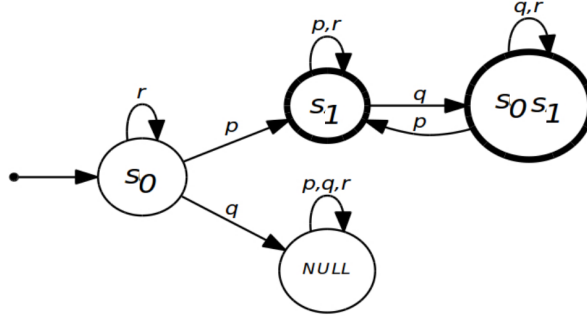


Figure 7.2: DFA Representation of the Distributed Security Model

Figure 7.2 (Ahmed et al., 2016) is the DFA representation of the distributed security model. The DFA can be represented as $M' = (Q', Z', R', q', F')$ where:

M' = the DFA

Q' = states of the DFA $\{s_0, s_1, s_0, s_1, NULL\}$

Z' = input symbols $\{r, p, q\}$

F' = acceptance state $\{s_1, s_0, s_1\}$

q' = initial state $\{s_0\}$

R' = transition $R' : Q' * Z'$ (see Matrix 7.2)

$NULL$ = a state of $\{s'_0, s'_1, s_0, s'_1\}$

This is a failure and from this state, there is no return.

$$\begin{pmatrix} R' & p & q & r \\ s_0 & s_1 & NULL & s_0 \\ s_1 & s_1 & s_0, s_1 & s_1 \\ s_0, s_1 & s_1 & s_0, s_1 & s_0, s_1 \\ NULL & NULL & NULL & NULL \end{pmatrix} \quad (7.2)$$

Matrix 7.2 is a 5×4 matrix and represents the transition function R' of the DFA, where:

Elements in column 0 represent the initial state of DFA $\{s_0, s_1, s_0, s_1, NULL\}$

Elements in row 0 represent the input symbols $\{p, q, r\}$.

The remainder of the elements represent the final state. For example, if s_0 receives an input p , then it moves to s_1 , which can be represented as $R(s_0, p) \rightarrow (s_1)$. Similarly, if s_1 receives an input q , then moves to (s_0, s_1) which can be represented as $R(s_1, q) \rightarrow (s_0, s_1)$.

7.2.2 Turing Machine

In this section, the Turing Machine representation of the security mechanism is discussed.

The Turing Machine can be represented as $M = (S, F, \delta, \varrho, q_0, \Sigma, \Gamma)$ where:

M = the Turing Machine

S = set of states of the Turing machine $\{s_0, s_1, s_2, s_3, s_4\}$

F = acceptance state $\{s_4\}$

δ = transition function and is the instruction set of Turing machine

$\{\delta : (S * \Gamma) \rightarrow (S * \Gamma * \{Left, Right\})$ (See Table 7.1 on the next page)

q_0 = initial state $\{s_0\}$

ϱ = a blank space on the tape $\{b\}$ where $\varrho \in \Gamma$ and $\varrho \notin \Sigma$

Σ = input alphabets $\{p, q\}$ where $\Sigma \in \Gamma$

Γ = tape elements $\{p, q, b, X, Y\}$

A Turing machine consists of a head and tape that contains infinite spaces to the right. The head starts scanning the left most alphabet of the tape, then traverses the entire tape and, on seeing a blank symbol, halts. The head of a Turing Machine can read, write and move left or right over the tape alphabets, with the condition that it can only move right in the first step. The Turing machine accepts language of the form $(p * n, q * n)$ (where p = addition of a component, q = deletion of an existing component); n should always be greater than or equal to two ($n \geq 2$).

Table 7.1: Instruction Set for Turing Machine

$\delta(s_0, p) \rightarrow (s_1, X, Right)$
$\delta(s_1, p) \rightarrow (s_1, p, Right)$
$\delta(s_1, q) \rightarrow (s_2, Y, Left)$
$\delta(s_2, p) \rightarrow (s_2, p, Left)$
$\delta(s_2, X) \rightarrow (s_0, X, Right)$
$\delta(s_1, Y) \rightarrow (s_1, Y, Right)$
$\delta(s_2, Y) \rightarrow (s_2, Y, Left)$
$\delta(s_0, Y) \rightarrow (s_3, Y, Right)$
$\delta(s_3, Y) \rightarrow (s_3, Y, Right)$
$\delta(s_3, b) \rightarrow (s_4, Halt, -)$

The instruction set in Table 7.1 (Ahmed et al., 2016) represents the working of the Turing Machine. Initially the head is at the left end of the tape and represents state s_0 . On reading an input symbol p on the tape, the Turing machine moves to state s_1 , writes X and moves right. On reading input p the Turing machine remains in state s_1 , writes p and moves right. On reading input q , moves to state q_2 , writes Y and moves left. On reading p on the tape, the Turing Machine remains in state s_2 and moves left. On reading X , it moves to the initial state s_0 and starts moving right. On reading, the tape element, Y , remains in state s_1 , then writes Y and moves right, similarly, if it is in state s_2 then it reads Y , remains in the same state and moves left. If the Turing Machine is in state s_0 and reads the tape alphabet Y , which means that all the input symbols p have been read by Turing Machine and moves to state s_3 . In state s_3 the Turing Machine remains in state s_3 , reads and writes the same symbol, until it reads a blank symbol $\{b\}$, then moves to state s_4 and halts.

Figure 7.3 on the next page (Ahmed et al., 2016) represents the Turing Machine which is formed by the instruction set as represented in Table 7.1. Furthermore, it accepts strings of the form $\{ppqq, pppqqq, pppppqqqq, pppppppqqqqq, \dots\}$ which belong to the language $(p * n, q * n)$.

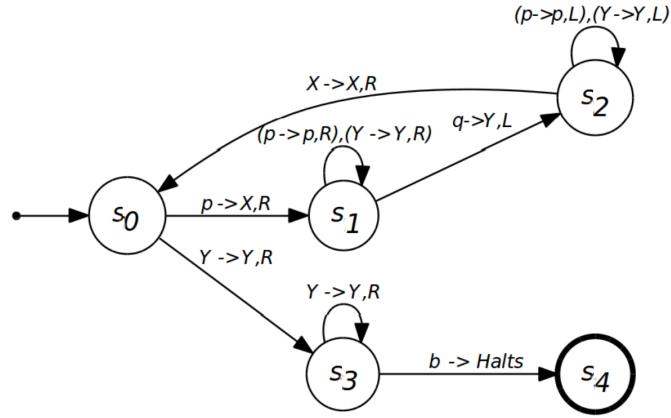


Figure 7.3: Turing Machine representation of the distributed security model

7.2.3 Security Mechanism Validation

A basic application is illustrated in this section. The application is considered to be deployed in Ki-Ngā-Kōpuku, and the illustration serves as the PoC for the security mechanism. The assumption is the need to develop an application that will take two numbers x and y as input from the user, add the numbers and return the result z back to the user.

The above activity of adding two numbers can be broken down into the following four steps:

1. Take two numbers as input from end-users
2. Interpret the processing request (perform addition in this case)
3. Add the numbers
4. Send the result back to the end-user

The first task is to design the application components. The application is divided into four components λ_1 (User Interface), λ_2 (Interpreter), λ_3 (Adder) and λ_4 (Deliverer). The naming of the components is insignificant and any name can be given to the components. The approach of componentisation is significant; the discussed approach is a possible one for the application. The components are described below:

λ_1 (User Interface): The user interface. It takes input from the user and displays output to the user. It is assumed to have a function $f(UI)$.

λ_2 (Interpreter): The interpreter or parser. It takes input from λ_1 , and decides what operation needs to be done on the input by interpreting user's request. It is assumed to have a function $f(interpret)$. The output from $f(UI)$ is input as parameter into $f(interpret)$.

λ_3 (Adder): The adder. It adds the number given to it. It is assumed to have a function $f(add)$. The output from $f(interpret)$ is input as parameter into $f(add)$.

λ_4 (Deliverer): The deliverer. It conveys the output back to the user interface (λ_1). It is assumed to have a function $f(deliver)$. The output from $f(add)$ is input as parameter into $f(deliver)$.

Figure 7.4 on the next page illustrates the sequence of processing for the above within Ki-Ngā-Kōpuku context.

The redundant copies of the components are not shown in Figure 7.4 on the following page for simplicity purpose. It is worth noting here that, in real world deployment, all the components of the application will have their redundant copies. So, for any reason, if a component, for example λ_3 (adder) becomes unavailable, Ki-Ngā-Kōpuku will ensure that there are replicas of λ_3 (adder) in the architecture. One of the replicas of λ_3 (adder) will become active in case λ_3 (adder) goes down. This is how Ki-Ngā-Kōpuku ensures no single point of failure through redundancy.

However, the sequence of processing progresses as follows:

1. End-user inputs two numbers x and y and asks the application to add them and then give the result back to the user.
2. λ_1 (User Interface) takes the numbers and the associated order and sends it to the λ_2 (Interpreter).

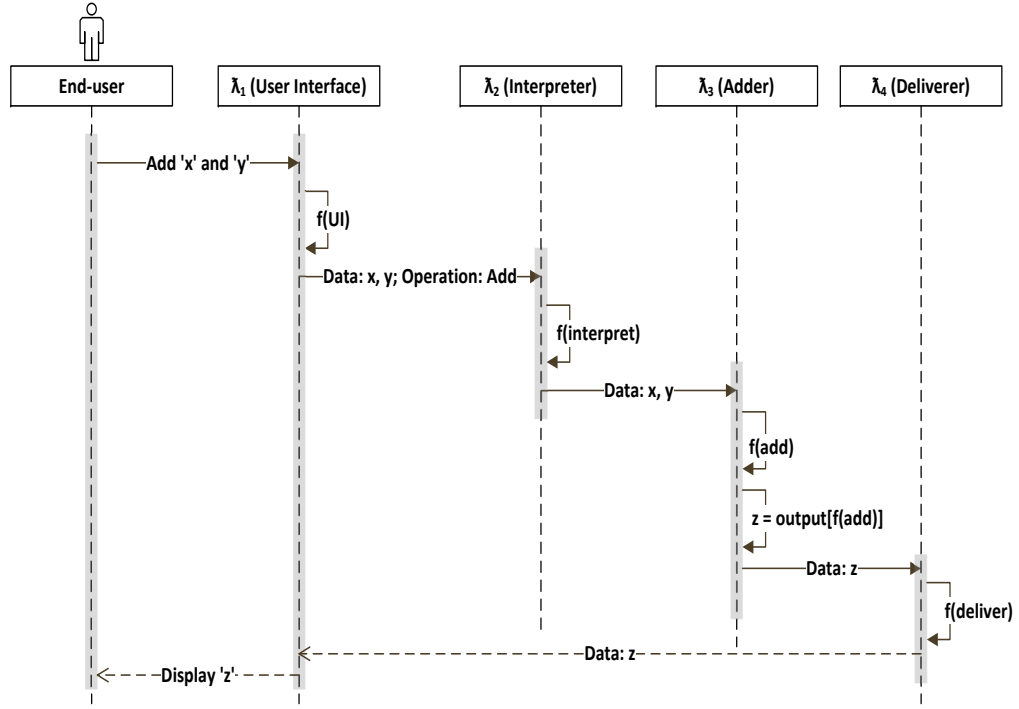


Figure 7.4: Proof of Concept of the Security Mechanism

3. λ_2 (Interpreter) analyses the input received from λ_1 (User Interface) and realizes that the numbers needs to be added which is a task to be performed by λ_3 (adder). λ_2 (Interpreter) forwards x and y to λ_3 (adder).
4. λ_3 (adder) adds x and y , and stores the result in z . λ_3 (adder) then forwards z to λ_4 (deliverer).
5. λ_4 (deliverer) conveys z to λ_1 (User Interface).
6. λ_1 (User Interface) displays the result to the end-user.

The above is based on ideal scenario where no unwanted events are assumed to be taking place. But in reality, it may not be the case. For example, the Cloud server holding an active component may go down. In such cases, a replica of that component will become active.

For the above scenario, the followings are assumed:

A component = λ .

Replica of a component = λ' .

An Application = P .

A function of the application = $p(f)$.

A function contained within a component = $\lambda[p(f)]$.

A function contained within a replica component = $\lambda'[p(f)]$.

Security mechanism = S .

Thus the security mechanism can be expressed as,

$$S \Rightarrow \exists S[\forall P[\forall \lambda[\lambda[p(f)]] \wedge \forall \lambda'[\lambda'[p(f)]]]]$$

7.2.4 Logical Simulation of Distribution

Referring to the Component Distribution mechanism as part of Ki-Ngā-Kōpuku security mechanism in Section 6.2 on page 127, a database Component Count is processed and shared with a newly added node. It is shown that the database is incremented by one for all the component distributed to the newly added node. However, it is not addressed before whether the database need to have its components count adjusted and updated in case a node goes down or die. This is answered in this section, which also establishes the validity for the relevant part of the security mechanism.

As illustrated in Section 6.2, to explain the security mechanisms, the only provision for the Component Count is to be incremented. It is irrelevant for the database to decrease the values of the components in a node that goes down.

It must be noted that the purpose of the Component Count database is not to hold the number of components that are alive within Ki-Ngā-Kōpuku architecture. The purpose of Component Count is to ensure that the distributions of the components are carried out in nearly balanced manner, so that one component is not distributed

repeatedly in such a way that it outweighs other components in total number of its replicas. It would not be an ideal situation where 500 replicas of a component exist, where the replicas for another component are, for example, 50 or so. If the number of existing replicas of a components are nearly balanced in number (proportionately distributed), then Ki-Ngā-Kōpuku architecture will have more powerful decentralised and distributed scenario that will be hard to compromise. To sum up, the purpose of Component Count database is not to indicate the number of existing components in Ki-Ngā-Kōpuku architecture, its sole purpose is to ensure nearly balanced distribution to subsequently bring robustness when it comes to redundancy with no single point of failure. The purpose of Component Count is explained below, which is a further validation on this aspect of Ki-Ngā-Kōpuku.

To ensure security, a Component Count database do not provide any information about the components, the origin of the components or any other parameter that may aid the traceability of the total architecture. As previously mentioned in Section 6.2, a Component Count database only contains total number of components an application is divided into, the number of components to be distributed, and the count of the components. It is assumed that an application is divided into three components λ_1 , λ_2 and λ_3 (*total_comp* attribute in the database). The application developers have decided that two components will be distributed whenever a distribution takes place (*dist_count* attribute in the database). It is also assumed that the current distribution is 1, 2, 1 respectively for λ_1 , λ_2 and λ_3 . Based on this assumption, a Component Count database may be a simple text file that may look like the one illustrated in Figure 7.5.

There may be other parameters that the application developers may find it necessary to include in the file (e.g. application identifier). However, the illustration in Figure 7.5 on the following page includes only the parameters that are bare minimum for the Component Count database, and is required to understand the concept and purpose of the database. Figure 7.5 on the next page depicts the very initial distribution scenario

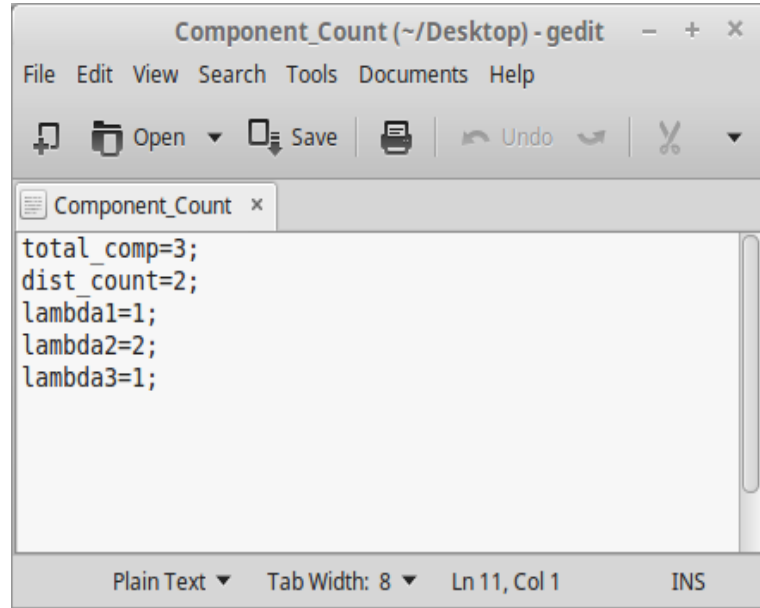


Figure 7.5: Component Count Database

that took place between two nodes (the information on the number of nodes is not in Component Count). This is the initial scenario from which the architecture will grow as new nodes are added and further distribution takes place. It may be assumed that Figure 7.5 is for the scenario where one node has the components λ_1, λ_2 ; and the other node has the components λ_2, λ_3 . This has resulted in the components count to be 1, 2, 1 respectively for λ_1, λ_2 and λ_3 .

According to the algorithm described in Section 6.2 on page 127, the components to be distributed to a new node are selected based on the information in Component Count database. The least numbered components are distributed. If the number is same for all components (e.g. $\lambda_1=1, \lambda_2=1, \lambda_3=1$), and two components are to be distributed, any of the two components are selected at random to be distributed.

Considering the above illustration, when the next new node comes alive, the existing nodes look at the Component Count which is $\lambda_1=1, \lambda_2=2, \lambda_3=1$. The least number of components are λ_1 and λ_3 here, and thus these are the two components that are to be distributed. Once the distribution is done, the component count becomes $\lambda_1=2, \lambda_2=2,$

$\lambda_3=2$, as λ_1 and λ_3 is distributed and their count is incremented by one. Now, for the next distribution, any of the two components may be chosen as they are of same count. Let us assume that that for the next distribution, λ_1 and λ_2 are randomly chosen for distribution. At the end of the distribution, the component count now becomes $\lambda_1=3$, $\lambda_2=3$, $\lambda_3=2$. For the next distribution, λ_3 (since it is the lowest count) and either of λ_1 or λ_2 will be distributed (since both of λ_1 and λ_2 have same count now). This cycle continues as more nodes are added to the architecture, and distribution keeps taking place. Based on the above scenario, Table 7.2 on the next page shows a logical simulation of first 20 distributions. The illustration assumes that the initial scenario was consisted of first two nodes node-1 and node-2 where node-1 contains components λ_1 , λ_2 ; and node-2 contains components λ_2 , λ_3 . This is the initial scenario for Ki-Ngā-Kōpuku to start functioning. Distribution starts when node-3 is added to the architecture. The addition of new nodes keeps going, so does the distribution, as illustrated in Table 7.2 on the following page.

Table 7.2 shows a pattern of distribution that can be predicted. However, this predictable pattern does not pose any security threats. This is due to the fact that the pattern of distribution tells only which components are distributed, no other information about the components are available from the Components Count. Besides, referring back to the security mechanism discussed in Section 6.2 on page 127, the processing in Ki-Ngā-Kōpuku is collectively carried out by a collection of nodes. Tampering with any single node would simply raise the alarm instead of opening a door for the attacker to proceed further to take over the whole architecture. And, without taking the whole architecture, an attacker will have little (if there is any at all) to do to achieve a successful attack or breach.

However, proceeding with the distribution by assuming a few nodes went down and at the same time, few new nodes are added to the architecture, the scenario depicted in Table 7.2 on the following page might look like something similar to the one illustrated

Table 7.2: Initial Distribution of an Application with Three Components

Node	Distribution No.	Component Count	Component Distributed
Node-1	Initial Scenario	$\lambda_1=1; \lambda_2=2; \lambda_3=1$	Initial Scenario
Node-2	Initial Scenario	$\lambda_1=1; \lambda_2=2; \lambda_3=1$	Initial Scenario
Node-3	1	$\lambda_1=2; \lambda_2=2; \lambda_3=2$	λ_1, λ_3
Node-4	2	$\lambda_1=3; \lambda_2=3; \lambda_3=2$	λ_1, λ_2
Node-5	3	$\lambda_1=3; \lambda_2=3; \lambda_3=3$	λ_1, λ_3
Node-6	4	$\lambda_1=4; \lambda_2=4; \lambda_3=3$	λ_1, λ_2
Node-7	5	$\lambda_1=4; \lambda_2=5; \lambda_3=4$	λ_2, λ_3
Node-8	6	$\lambda_1=5; \lambda_2=5; \lambda_3=5$	λ_1, λ_3
Node-9	7	$\lambda_1=6; \lambda_2=6; \lambda_3=5$	λ_1, λ_2
Node-10	8	$\lambda_1=6; \lambda_2=7; \lambda_3=6$	λ_2, λ_3
Node-11	9	$\lambda_1=7; \lambda_2=7; \lambda_3=7$	λ_1, λ_3
Node-12	10	$\lambda_1=8; \lambda_2=8; \lambda_3=7$	λ_1, λ_2
Node-13	11	$\lambda_1=8; \lambda_2=9; \lambda_3=8$	λ_2, λ_3
Node-14	12	$\lambda_1=9; \lambda_2=9; \lambda_3=9$	λ_1, λ_3
Node-15	13	$\lambda_1=10; \lambda_2=10; \lambda_3=9$	λ_1, λ_2
Node-16	14	$\lambda_1=10; \lambda_2=11; \lambda_3=10$	λ_2, λ_3
Node-17	15	$\lambda_1=11; \lambda_2=11; \lambda_3=11$	λ_1, λ_3
Node-18	16	$\lambda_1=12; \lambda_2=12; \lambda_3=11$	λ_1, λ_2
Node-19	17	$\lambda_1=13; \lambda_2=12; \lambda_3=12$	λ_1, λ_3
Node-20	18	$\lambda_1=13; \lambda_2=13; \lambda_3=13$	λ_2, λ_3
Node-21	19	$\lambda_1=14; \lambda_2=14; \lambda_3=13$	λ_1, λ_2
Node-22	20	$\lambda_1=14; \lambda_2=14; \lambda_3=13$	none

in Table 7.3 on the next page.

Examination of Table 7.3 reveals that a number of nodes from the architecture went down and unavailable. These are nodes 4, 6, 9, 12, 15 and 18. At the same time, a number of new nodes are added to the architecture. The newly added nodes are node 21 to 29 where distributions have taken place in the way illustrated before. The component count did not decrease when a node went off and became unavailable. Table 7.3 suggests that the architecture always moves towards a nearly balanced distribution without decreasing the components count in the case of a node going down.

The above simulation establishes that the component count does not reveal any

Table 7.3: Changed Distribution Scenario with no Decrement in Component Count

Node	Distribution No.	Component Count	Component Distributed
Node-1	Initial Scenario	$\lambda_1=1; \lambda_2=2; \lambda_3=1$	Initial Scenario
Node-2	Initial Scenario	$\lambda_1=1; \lambda_2=2; \lambda_3=1$	Initial Scenario
Node-3	1	$\lambda_1=2; \lambda_2=2; \lambda_3=2$	λ_1, λ_3
*	*	*	*
Node-5	3	$\lambda_1=3; \lambda_2=3; \lambda_3=3$	λ_1, λ_3
*	*	*	*
Node-7	5	$\lambda_1=4; \lambda_2=5; \lambda_3=4$	λ_2, λ_3
Node-8	6	$\lambda_1=5; \lambda_2=5; \lambda_3=5$	λ_1, λ_3
*	*	*	*
Node-10	8	$\lambda_1=6; \lambda_2=7; \lambda_3=6$	λ_2, λ_3
Node-11	9	$\lambda_1=7; \lambda_2=7; \lambda_3=7$	λ_1, λ_3
*	*	*	*
Node-13	11	$\lambda_1=8; \lambda_2=9; \lambda_3=8$	λ_2, λ_3
Node-14	12	$\lambda_1=9; \lambda_2=9; \lambda_3=9$	λ_1, λ_3
*	*	*	*
Node-16	14	$\lambda_1=10; \lambda_2=11; \lambda_3=10$	λ_2, λ_3
Node-17	15	$\lambda_1=11; \lambda_2=11; \lambda_3=11$	λ_1, λ_3
*	*	*	*
Node-19	17	$\lambda_1=13; \lambda_2=12; \lambda_3=12$	λ_1, λ_3
Node-20	18	$\lambda_1=13; \lambda_2=13; \lambda_3=13$	λ_2, λ_3
*	*	*	*
Node-22	20	$\lambda_1=14; \lambda_2=14; \lambda_3=13$	none
Node-23	21	$\lambda_1=15; \lambda_2=14; \lambda_3=14$	λ_1, λ_3
Node-24	22	$\lambda_1=15; \lambda_2=15; \lambda_3=15$	λ_2, λ_3
Node-25	23	$\lambda_1=16; \lambda_2=16; \lambda_3=15$	λ_1, λ_2
Node-26	24	$\lambda_1=16; \lambda_2=17; \lambda_3=16$	λ_2, λ_3
Node-27	25	$\lambda_1=17; \lambda_2=17; \lambda_3=17$	λ_1, λ_3
Node-28	26	$\lambda_1=18; \lambda_2=18; \lambda_3=17$	λ_1, λ_2
Node-29	27	$\lambda_1=18; \lambda_2=19; \lambda_3=18$	λ_2, λ_3
Node-30	28	$\lambda_1=19; \lambda_2=19; \lambda_3=19$	λ_1, λ_3
Node-31	29	$\lambda_1=20; \lambda_2=20; \lambda_3=19$	λ_1, λ_2

information that might be of security concern. When few nodes go down, the component count does not reflect the actual number of nodes existent in the architecture. This is the case in the scenario in Table 7.3. Thus, the Component Count, in fact, may give an attacker confusing information about the architecture that will inherently further

integrate security within Ki-Ngā-Kōpuku architecture. The above validates the relevant part of the security mechanism.

7.3 Further Validation

The following discussion explores the suitability of Ki-Ngā-Kōpuku by exploring a few recent Cloud and data breaches and how Ki-Ngā-Kōpuku may help to prevent those breaches. This further validates Ki-Ngā-Kōpuku as a security mechanism.

7.3.1 WannaCry

As the writing is in progress, one of the biggest malware attack in history has struck and affected many countries. It is codenamed ‘WannaCry’. It is a ransomware (NZHerald, 2014) used in large scale attacks (CERT, 2017). A ransomware is malicious software that typically encrypts someone’s resources to make it unreadable and threatens the owner of the data in various ways. The threats could be to lock data and then delete it if money is not paid. Additionally, hackers may also threaten to publish sensitive hacked information if a ransom is not paid. WannaCry takes the first approach. It compromises users’ computing devices, encrypts their data to make it unreadable, and asks the victim to pay a ransom to recover data (Symantec, 2017).

Further research on Ki-Ngā-Kōpuku in Appendix B on page 211 notes that the future development of Ki-Ngā-Kōpuku will take data security into account. The security of data will be provided using the same theme of having redundant copies of data, stored in secured (encrypted) format. If an application is deployed using Ki-Ngā-Kōpuku, the users data will reside in different Cloud servers. A user can have the image of their own personal computer stored on the Cloud within Ki-Ngā-Kōpuku architecture. At this point, an attack like WannaCry may not give the attacker ability to read the user’s data. The attack may be successful to ‘lock’ users’ data, but the user will not need to

recover the locked data, let alone pay a ransom. This is due to the fact that the users data are deployed within Ki-Ngā-Kōpuku architecture and thus there are redundant copies. The user may simply discard the locked data and restore a new redundant image of their computing device provided by the CSP who is using Ki-Ngā-Kōpuku architecture to provide their service. The componentisation of an application and data, having redundant copy with no single point of failure, and the self-healing capability of Ki-Ngā-Kōpuku may make attacks like WannaCry powerless.

7.3.2 Mirai Botnets

Wired (2017) reports the strengths of Botnets may be used by attackers to carry out DDoS attack, that may be used to reignite attacks like WannaCry. As described in (TheRegister, 2017), the Mirai botnet or its variant can be very powerful to carry out successful DDoS attack. It is previously discussed that Ki-Ngā-Kōpuku can successfully combat DoS and DDoS attacks. Any attacks based on botnets can be effectively mitigated by Ki-Ngā-Kōpuku. This can be achieved simply by switching the activity from a compromised node to a replica node without making the attacker aware of this. Thus, while successfully bypassing botnet and DDoS attacks, opportunities exist for Ki-Ngā-Kōpuku to evolve as an effective on-demand honey-pot, as Ki-Ngā-Kōpuku may acquire the ability to turn itself into a honey-pot if an attack takes place.

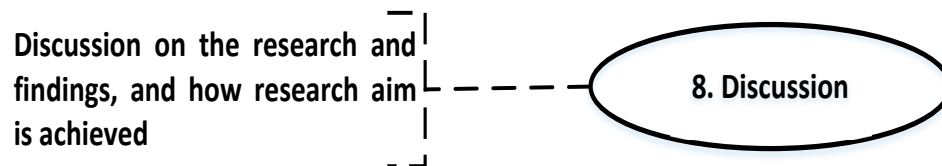
7.4 Conclusion

The validation of the system discussed in this chapter completes the discussion on Ki-Ngā-Kōpuku blueprint. The validation shows that deployment of Ki-Ngā-Kōpuku is achievable. If an application is deployed using Ki-Ngā-Kōpuku, the application is componentised that does not have a single point of failure as the components have their redundant copies randomly scattered across the architecture. It is now time to look

back into the research design presented in Chapter 3 on page 80, to map the conducted research to the decided research approach. This will help to establish how the research satisfies the criteria and requirements set out in the planned research method. This, along with few other factors, are addressed in Chapter 8 on the next page.

Chapter 8

Discussion



8.1 Introduction

This chapter outlines how the research conforms to the research design discussed earlier in Chapter 3. The discussion reflects on what is done as part of research, and whether the expectations are met. It also identifies how the presented research deals with the hypothesis. The mapping to the research design and the conducted research show how the research activities are carried out in the planned way to achieve the research aim. This is presented in Section 8.2 on the following page of this chapter.

The provision of using other technologies is also considered in the discussion. In Section 8.3 on page 181, it analyses whether Ki-Ngā-Kōpuku can be used with other existing security tools and technologies. It also explores how using any existing technology may complement Ki-Ngā-Kōpuku.

8.2 Research and Research Design: Mapping

In this section, the research questions and hypotheses along with the hypothesis testing methods are discussed. A mapping of these to the research is shown below to identify which part of the research is associated with any specific research question or hypothesis. The discussion also confirms that the research aim is achieved by following the planned research design. The discussion in this section refers back to the mapping presented in Figure 3.2 on page 95 to establish the link between the research plan and the conducted research. As discussed in Sections 3.2.1 on page 82 and 3.3 on page 88, three research questions are planned to be answered through four hypotheses, and the hypotheses are planned to be tested by five testing methods. The following discussion presents the examples from previous chapters that show how this is carried out in the research. Sections 8.2.1, 8.2.2 on page 167 and 8.2.3 on page 170 show how the research achievements progressed and the answers to RQs sought. Section 8.2.4 on page 178 presents a critical analysis on the conducted research.

8.2.1 Research Question 1

Figure 8.1 shows the mapping RQ1 and the hypothesis, as well as the methods used to test the hypotheses.

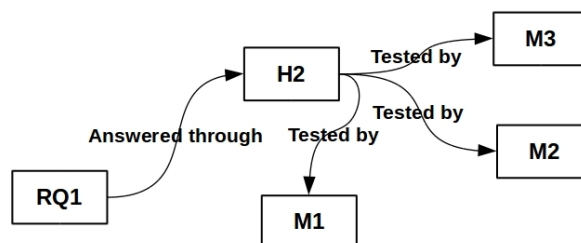


Figure 8.1: Mapping RQ-1, Hypothesis and Testing Methods

RQ1, H2, M1, M2 and M3 are re-stated below:

RQ1: What are the contexts from which a Cloud security breach may emerge?

- H2:** Threats to CC are unique when compared to other computing models.
- M1:** Explore historical evidence to understand the implications and limitations of different Cloud security mechanisms.
- M2:** Develop the theoretical model to define the distributed security model. The principles of DSR approach will be employed in this method. Formal methods within the context of the principles of DSR approach will be employed in this method.
- M3:** Explore approaches of different processing aspects of a distributed security model, and determine best suited algorithm for the processing.

Achieving plan in Figure 8.1 on the preceding page:

The following discussion shows how the accomplishments of the testing methods are reflected in the research, then how such accomplishments lead to testing the hypothesis. This is eventually mapped to the research question to show how it is answered in the research.

M1: Extensive literature review is done and presented in Chapter 2. This secondary research helps to understand state-of-the-art Cloud security. Along with literature review, a number of recent Cloud breach-related case analyses help to understand the processes of Cloud security breaches. Secondary research of CC security is presented in Section 2.3 on page 35. Section 2.4 on page 38 helps to understand the major security points in a Cloud infrastructure. This lead to developing a threat taxonomy for CC which is presented in Section 2.4.3 on page 44. Section 2.5 on page 64 presents discussion and findings on current Cloud security models. This helps with understanding how current research is addressing Cloud security concerns which, in turn, help with understanding the processes of Cloud security breach. The relevant discussion also helps with understanding where Cloud security is aligned in traditional computer network

security.

M2: SRS for Ki-Ngā-Kōpuku are developed and discussed in Section 4.3.1 on page 99. The artefacts (deliverables) from Chapter 4 are Problem Definition, Problem Specifications, the SRS (both requirement perspective and framework perspective). It is proposed that a security model consists of an RA and its underlying security mechanism. The conceptual view of the security model is presented in Section 5.2 on page 113. Here, Formal Expressions are used to define the concept of the decentralised, distributed security model. The RA and the overview of the security mechanism are presented in Section 5.3 on page 119. The artefacts (deliverables) from Chapter 5 are the conceptual view of the security model, the RA and the high-level view of the security mechanism for the security model. The development of the above artefacts prompted exploration to gain an understanding of the processes of Cloud security breaches.

M3: The algorithms for Ki-Ngā-Kōpuku are presented in the form of flowcharts in Section 6.2 on page 127. The discussion presents overall logical constructs of the security mechanism as well as all the processing steps and algorithms Ki-Ngā-Kōpuku requires as part of its security mechanism. The development of the security mechanism is aided by the findings and understanding of the processes of Cloud security breaches. Understanding the processes in Cloud security breach contribute to the formulation of the specifics of the algorithms and processing steps.

H2: Based on the findings through secondary research, a threat taxonomy for CC is proposed. The threat taxonomy and the literature review shows that CC inherits security concerns from all computing means including a networked computing scenario. This has led to arguing in this research that CC is an encapsulating wrapper for all other computing means and thus CC inherits security concerns from all other computing means including networked computing scenarios. Thus, the outcome of the research disproves H2, by concluding that the security threats in a networked computing environment pose similar threats to those of CC security threats. Thus, threats to CC are not unique when

compared to other computing models.

RQ1: Work on H2 helps to understand different contexts of CC, Cloud security and to develop a threat taxonomy for CC, as discussed in Section 2.4 on page 38. The work also leads to the realisation that threats to CC are not unique when compared to other computing models. As well, understanding the threats and Cloud security also helps to develop specifics of the security mechanism as discussed above. These constructs an understanding of the contexts from which a security breach in CC may emerge. Exploring the answer to RQ1 is complemented by the hypothesis and the testing methods.

8.2.2 Research Question 2

Figure 8.2 shows the mapping RQ2 and the hypotheses, as well as the methods that are used to test the hypotheses.

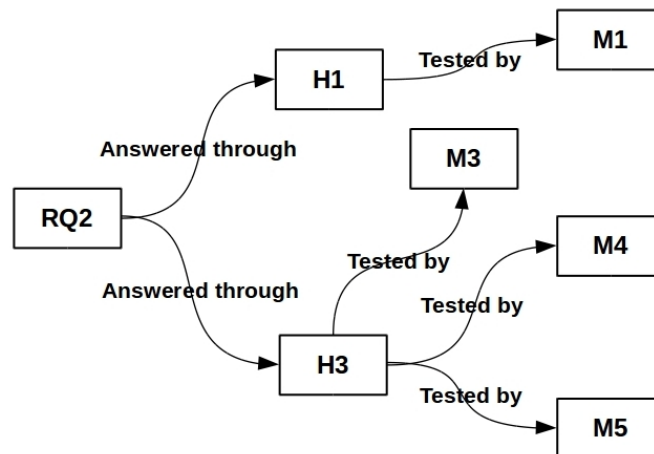


Figure 8.2: Mapping RQ-2, Hypotheses and Testing Methods

RQ2, H1, H3, M1, M3, M4 and M5 are re-stated below:

RQ2: What measures can be applied that avoid a single point of failure in Cloud-based systems?

- H1:** Existing software architecture provide the conditions for a single point of failure.
- H3:** In a decentralised and distributed architecture, the problem of a single point of failure can be eliminated.
- M1:** Explore historical evidence to understand the implications and limitations of different Cloud security mechanisms.
- M3:** Explore approaches of different processing aspects of a distributed security model, and determine best suited algorithm for the processing.
- M4:** Explore approaches of how an application can be self-healing and can uninterruptedly continue to serve even in the case of a breach or attack, and determine best suited algorithm for this. Formal methods within the context of the principles of DSR approach will be employed in this approach.
- M5:** Explore approaches of how a logical Proof of Concept (PoC) can be developed, and ways to validate various aspects of the distributed and decentralised security mechanism; and determine the impact and integrity between Cloud architecture and security model. Formal methods and logical reasoning for system validity within the context of DSR will be employed in this method.

Achieving plan in Figure 8.2 on the previous page:

The following discussion shows how the accomplishments of the testing methods are reflected in the research, then how this leads to testing the hypothesis. This is eventually mapped to the research question to show how it is answered in the research.

M1: Extensive literature review is done and presented in Chapter 2. This secondary research helps to understand state-of-the-art Cloud security. Secondary research of CC security, presented in Section 2.3 on page 35, helps to understand the current state of security for CC. Section 2.4 on page 38 helps in understanding the major security

points in a Cloud infrastructure. Section 2.6 on page 73 explores the implications of having a security model in distributed form. The findings in the sections suggest that the current state of security in the Cloud does not address the aspects of distributed security mechanism to a great extent, despite the concern and opinion that the distributed nature of resources in CC reveal more attack vectors and thus a distributed security approach is more suitable for CC.

H1: The findings as discussed in M1 above imply that, Cloud resources are distributed. It is also found that the resource distribution aspect of CC makes it complex and allures the attackers at the same time. The findings also suggests that the existing security models do not greatly address the provision of employing a distributed security model. It is also found that researchers expressed their concerns about the current state of security in the Cloud, and opinions exist that a distributed security model could be more suitable to Cloud security due to its nature of having resources distributed. The findings thus disprove H1 by establishing the fact that the distributed nature of a security mechanism may have an impact on CC which is distributed in nature. This leads to the realisation that existing software architecture associate with the conditions for a single point of failure.

M3: Section 6.2 on page 127 illustrates the processing details and algorithms outlining the processing details of the security model.

M4: Section 6.2.4 on page 136 shows how a node can work with another node, and in case a node it compromised (or goes down), how a substitute node may become active and thus the system would exhibit its self-healing feature. Artefacts produced as part of this method are the set of processing steps and algorithms for Ki-Ngā-Kōpuku.

M5: The developed decentralised and distributed security model with self-healing mechanism is validated using formal methods (in Section 7.2.1 on page 146), showing that the security mechanism is Turing complete (in Section 7.2.2 on page 149), using illustrative example of an application logically deployed using Ki-Ngā-Kōpuku (in

Section 7.2.3 on page 151), and by means of logical simulation of part of the security mechanism (in Section 7.2.4 on page 154) to show the validity of the relevant part of the security mechanism.

H3: The discussions related to M3, M4 and M5 above show that the proposed distributed security model employs an exception monitoring mechanism as an inherent part of its processing steps. By using a redundant approach and eliminating single point of failure, Ki-Ngā-Kōpuku generates exception at the very moment a suspicious event is indicated, which is possible due to the decentralised and distributed nature of the proposed security model. The self healing nature of the security model is evident, as a compromised node can easily be substituted by its corresponding replica node. Thus, in a decentralised and distributed architecture, the problem of a single point of failure can be eliminated. This validates H3 and implies that the distributed security model can instantly remedy an attack.

RQ2: The research disproves H1 by proving that a distributed security model is more suitable for CC, as CC itself distributes resources. It also implies that the existing software architecture associate with the conditions for a single point of failure. H3 is validated by the research as discussed above. This implies that a decentralised and distributed security model enhances Cloud security. This, at the same time, also eliminates single point of failure and minimises the lack of resilience in CC. The research shows that if an application is divided into several components to randomly scatter among random Cloud servers to have architecture with redundancy with no single point of failure, the security in the Cloud may be enhanced.

8.2.3 Research Question 3

Figure 8.3 on the next page shows the mapping RQ3 and the hypothesis, as well as the methods that are used to test the hypotheses.

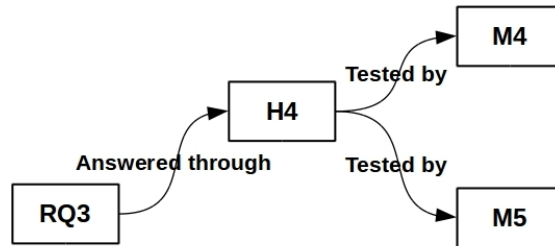


Figure 8.3: Mapping RQ-3, Hypothesis and Testing Methods

RQ3, H4, M4 and M5 are re-stated below:

RQ3: How the loss of availability of Cloud services can be minimised?

H4: To minimise service unavailability and to improve CC security, the design strategy of a security mechanism is important.

M4: Explore approaches of how an application can be self-healing and can uninterruptedly continue to serve even in the case of a breach or attack, and determine best suited algorithm for this. Formal methods within the context of the principles of DSR approach will be employed in this approach.

M5: Explore approaches of how a logical Proof of Concept (PoC) can be developed, and ways to validate various aspects of the distributed and decentralised security mechanism; and determine the impact and integrity between Cloud architecture and security model. Formal methods and logical reasoning for system validity within the context of DSR will be employed in this method.

Achieving plan in Figure 8.3:

The following discussion shows how the accomplishments of the testing methods are reflected in the research, then how this leads to testing the hypothesis. This is eventually mapped to the research question to show how it is answered in the research.

M4: Section 6.2.4 on page 136 shows how a node can work with another node, and in case a node is compromised (or goes down), how a substitute node may become

active and thus the system would exhibit its self-healing feature. Artefacts produced as part of this method are the set of processing steps and algorithms for Ki-Ngā-Kōpuku.

M5: The developed decentralised and distributed security model with self-healing mechanism is validated using formal methods (in Section 7.2.1 on page 146), showing that the security mechanism is Turing complete (in Section 7.2.2 on page 149), using an illustrative example of an application logically deployed using Ki-Ngā-Kōpuku (in Section 7.2.3 on page 151), and by means of logical simulation of part of the security mechanism (in Section 7.2.4 on page 154) to show the validity of the relevant part of the security mechanism.

H4: The works and validation carried out and described in M4 and M5 above inform that the decentralised and distributed nature of Ki-Ngā-Kōpuku can be an effective tool against the threats it is envisioned to combat. Research shows that the decentralisation and distribution of a security mechanism by means of application componentisation and redundancy also helps to achieve a self-healing quality. This subsequently maximises service availability. Thus the research validates H4 in that, a distributed security model is an effective tool for providing security for CC, and choosing a distributed and decentralised design strategy strengthen security for CC to maximise service availability. The design strategy of a security mechanism is important to improve CC security and to minimise service unavailability.

RQ3: RQ3 is answered by validating H4 through methods M4 and M5. The discussions under M4 and M5 above point to parts of the research, where the approaches to employ the proposed decentralised and distributed security model are presented. The research finds that the componentisation of an application and randomly scattering the components into a random number of scattered nodes among random Cloud servers may evolve to effectively deploy a decentralised and distributed security model for CC. Such a system prevents service unavailability through distributed redundancy and elimination of single point of failure. The design strategy of a security mechanism is

important to improve CC security and to minimise service unavailability.

The above evidence of research achievements are summarised in Table 8.1.

Table 8.1: Evidence of Research Achievements

RQ	Hypothesis	Research Evidence	Artefacts/Deliverables
RQ1	H2–Disproved	<ol style="list-style-type: none"> 1. Disproved by testing methods M1, M2 and M3. 2. Section 2.3 on page 35 that helps to understand the current state of security for CC. (M1) 3. Section 2.4 on page 38 helps to understand the major security points in a Cloud infrastructure. (M1) 4. A threat taxonomy for CC in Section 2.4.3 on page 44. (M1) 	<ol style="list-style-type: none"> 1. Literature Review that shows current security state as well as existing security models in CC. 2. A generalised threat taxonomy for CC. 3. Problem Definition 4. Problem Specification
		<ol style="list-style-type: none"> 5. Section 2.5 on page 64 presents discussion and findings on current Cloud security models. (M1) 6. SRS for Ki-Ngā-Kōpuku in Section 4.3 on page 98. (M2) 7. Conceptual view of the security model in Section 5.2 on page 113. (M2) 8. The RA and the overview of the security mechanism in Section 5.3 on page 119. (M2) 9. Processing steps and algorithms for Ki-Ngā-Kōpuku in Section 6.2 on page 127. 	<ol style="list-style-type: none"> 5. SRS 6. Conceptual View of the security model. 7. Reference Architecture 8. Security mechanism including its processing steps and algorithms.

Continued on next page

Table 8.1 – Continued from previous page

RQ	Hypothesis	Research Evidence	Artefacts/Deliverables
RQ2	H1–Disproved H3–Validated	<ol style="list-style-type: none"> 1. Disproved by testing methods M1, M3, M4 and M5. 2. Section 2.3 on page 35 that helps to understand the current state of security in CC. (M1) 3. Section 2.4 on page 38 helps to understand the major security points in a Cloud infrastructure. (M1) 4. Section 2.6 on page 73 explores the implications of having a security model in distributed form. (M1) 5. Processing steps and algorithms for Ki-Ngā-Kōpuku in Section 6.2 on page 127. (M3) 	<ol style="list-style-type: none"> 1. Literature review to understand current state of Cloud Security, major security points in Cloud infrastructure, and implications of distributed security model for CC. 2. Security mechanism including its processing steps and algorithms. 3. Validation of the security model.

Continued on next page

Table 8.1 – Continued from previous page

RQ	Hypothesis	Research Evidence	Artefacts/Deliverables
		<p>6. Section 6.2.4 on page 136 shows how a node can work with another node, and in case a node it compromised (or goes down), how a substitute node may become active and thus the system would exhibit its self-healing feature. (M4)</p> <p>7. Security model with self-healing mechanism is validated using formal methods (Section 7.2.1 on page 146). (M5)</p> <p>8. Section 7.2.2 on page 149 shows that the security mechanism is Turing complete. (M5)</p> <p>9. Illustrative example of an application logically deployed using Ki-Ngā-Kōpuku in Section 7.2.3 on page 151. (M5)</p>	
		<p>10. Logical simulation of part of the security mechanism in Section 7.2.4 on page 154. (M5)</p>	

Continued on next page

Table 8.1 – Continued from previous page

RQ	Hypothesis	Research Evidence	Artefacts/Deliverables
RQ3	H4–Validated	<ol style="list-style-type: none"> 1. Section 6.2.4 on page 136 shows how a node can work with another node, and in case a node it compromised (or goes down), how a substitute node may become active and thus the system would exhibit its self-healing feature. (M4) 2. Security model with self-healing mechanism is validated using formal methods (Section 7.2.1 on page 146). (M5) 	<ol style="list-style-type: none"> 1. Security mechanism including its processing steps and algorithms. 2. Validation of the security model.
		<ol style="list-style-type: none"> 3. Section 7.2.2 on page 149-7.2.2 shows that the security mechanism is Turing complete . (M5) 4. Illustrative example of an application logically deployed using Ki-Ngā-Kōpuku in Section 7.2.3 on page 151. (M5) 5. Logical simulation of part of the security mechanism in Section 7.2.4 on page 154. (M5) 	

8.2.4 Reflection on Research Achievements

The progress of the research through hypothesis testing approaches to prove or disprove the hypothesis lead to a number of realisations, as discussed below:

M1: The currently existing security models for CC do not exhibit distributed aspects in their entirety. The terms Security Model or Security Mechanism has no standard distinguishing definitions, and are often used interchangeably. The research proposes definitions for them by clearly outlining the distinction between them. While a number of security models exist for CC, and some of the security models take the distributed aspects into account to some extent; a truly distributed and decentralised security model could not be found. Ki-Ngā-Kōpuku fill this gap and adds a truly distributed and decentralised security model to the knowledge base. The absence of a truly distributed approach towards Cloud security may help CC to loose its appeal. The implications of Cloud security (or lack of it) is massive that is supported by big Cloud breaches. It is found that limitations of existing Cloud security mechanisms is one driving factor behind such breaches. As established through literature review, Cloud itself is distributed, so should its security mechanism be, to protect Cloud better.

It is also found that CC encapsulates all computing means and thus threats to all other computing means and approaches are applicable to CC too. This, along with the analysis of Cloud specific threats act as the motivation to propose a threat taxonomy (in Section 2.4.3 on page 44) for CC that is another innovative outcome of the research, and an addition to the knowledge base. CC incorporated management, security and storage aspects. One of the realisations that the taxonomy suggests is that, threats for CC may emerge both from technological and human factors. This indicates that security concerns in Cloud is heterogeneous and a single security solution for all security concerns in CC may be overambitious. The state-of-the-art of Cloud security is claimed to be unclear and a distributed approach to Cloud security is sought – a motivation to

propose Ki-Ngā-Kōpuku.

M2, M3, M4, M5: To develop the processing steps for the security mechanism, it is found that the aspects of component distribution and redundancy adds complexity to system design in terms of performance overheads. It is also realised that the possibility of reverse engineering might be a sensitive loophole in designing distributed and decentralised systems. Any kind of tractability in a system's processing may pose threat of reverse engineering, this is a reason for which the Component Count in Ki-Ngā-Kōpuku is not updated when a node goes down, to enforce barriers to prohibit reverse engineering.

One of the challenges of designing a decentralised and distributed system is to define the boundary of the system, and to decide the extent to which the system should expand. In the case of Ki-Ngā-Kōpuku, this is confined by introducing the concept of BEC and FEC. It is envisioned that Ki-Ngā-Kōpuku will evolve to work and spread its nodes in an open cyber-space (i.e. not confined within any infrastructural boundary, for example, BEC).

Making a system decentralised and distribute incorporates processing and performance overheads. Taking the Ki-Ngā-Kōpuku context into account, a crucial question to solve would be determining total number of nodes that participates in a processing. Increasing the number of nodes ensures better reliability of the system goal, but it demands resource and performance overheads. Thus, it becomes crucial to decide the factors that can be compromised while designing a decentralised and distributed system.

Existing and available technology is a factor that might influence and bias the design decision of a system. The proposed security model is not developed to ensure that it is tailored to any existing technology. The research rather focused on what can be achieved logically and computationally. As this is the case for Ki-Ngā-Kōpuku, technological provisions can be developed for the security model, in case the existing technologies are unable to cater for the deployment of Ki-Ngā-Kōpuku.

The processes in the security mechanism of Ki-Ngā-Kōpuku based on the principle of collaborative processing. In collaborative processing, the approach needs to be such that a group of nodes will work together without having knowledge on the total context (i.e. certain particulars about other nodes), and without having the opportunity to team up with other nodes to form hidden sub-groups within the group of nodes. This principle is reflected in the processing steps of Ki-Ngā-Kōpuku discussed in Section 6.2 on page 127.

To sum up, exploring answers to research problems reveal that CC is not a distinct technology, it is rather a conceptual wrapper for any computing means. This implies that threats to CC is greater than the sum of all threats in all other computing means. As a result, the importance of security and research to further strengthen CC security is of importance. Human factors play an important role in initiating breaches. Technological factors are mostly structured and thus predictable. Human factors are mostly unstructured and thus unpredictable. For this, a security solution that prevents or at least have the ability to overcome human factor related breaches qualifies to be considered properly. Through redundancy, decentralisation and elimination of single point of failure, Ki-Ngā-Kōpuku can overcome some human factor related breaches. Let us consider a scenario as an example: a ransomware encrypts data and asks for ransom to have the data decrypted. For such scenario, using Ki-Ngā-Kōpuku would be remedial in that, the redundant copies of the data enables the victim to simply ignore the attackers demand of ransom as the victim has further copies of the data held by the attacker.

A single point of failure for a system can be avoided using either of the two approaches: make multiple copies of the whole system, or create multiple copies of part of the system and scatter them into different servers that results in having no single repository for the whole system. Ki-Ngā-Kōpuku adopts the latter approach which is more complex to predict compared to the former one. Subsequently, the context or

overall behaviour as well as characteristics of a system deployed using Ki-Ngā-Kōpuku becomes complex to map for an attacker, resulting in increase security and better resilience. The approach used in Ki-Ngā-Kōpuku eliminated single point of failure at the same time that eventually leads to maximised service availability compared to other approaches to deploy Cloud services.

8.3 Complementing other Technologies

Chapter 6 shows that part of the processing steps of Ki-Ngā-Kōpuku uses other well-defined and proved technologies. The uses of encryption, checksum generation or hashing algorithms are such examples. The provision to use these technologies further validates the proposed security mode. Ki-Ngā-Kōpuku is able to complement other technologies to validate itself and to use the capability of other technologies to enhance its capability. The following discussion is an analysis of a few technologies and how they may complement Ki-Ngā-Kōpuku.

8.3.1 Blockchain and Ki-Ngā-Kōpuku

Blockchain technology uses a decentralised and distributed ledger for verification and recording of transactions (Sevres & Kakavand, 2017). To explore how BlockChain may help Ki-Ngā-Kōpuku, first a quick review or overview of Ki-Ngā-Kōpuku is presented. Ki-Ngā-Kōpuku is a decentralised, distributed security model for CC that divides an application into several components, and scatters these components into random number of nodes. A Ki-Ngā-Kōpuku node is a logical wrapper that holds random components of an application. The nodes reside in Cloud servers. A Cloud server may contain a random number of nodes. In Ki-Ngā-Kōpuku, any existing node may go down or die at any moment, and a new node may come alive. A new node is an empty Ki-Ngā-Kōpuku node only with standard Ki-Ngā-Kōpuku functions, but with

no application components. Thus, when a new node comes alive, other existing nodes distribute application components to the new node. For this, Ki-Ngā-Kōpuku keeps track of the number of components distributed (but no further information that may aid an attacker to build a mapping of nodes).

The above is the current research on Ki-Ngā-Kōpuku, and it aims to maximise application availability. It also envisions successfully combatting against cyber attacks like DoS or DDoS attack. As part of future development, data security is also planned to be included as a function of Ki-Ngā-Kōpuku. Primary research on this has shown that trusted sources (nodes or Cloud servers) and the validity of trusted source will be important for data security and integrity.

For the above scenario of current and future planned development of Ki-Ngā-Kōpuku, blockchain technology may help significantly. The validity of the component count can be verified by using blockchain. Integrating a validation method like blockchain would minimise the current processing overhead for Ki-Ngā-Kōpuku and would increase overall performance. As well, in future development when data security is taken into consideration, using blockchain would help to identify integrated trusted sources by minimising current processing overheads associated with Ki-Ngā-Kōpuku, further optimising for real-time processing. Thus, using blockchain technology would complement Ki-Ngā-Kōpuku by eliminating its processing overheads and subsequently making it suitable for any kind of application including applications involving real-time processing and transactions.

8.3.2 Machine Learning

As defined by Hall, Dean, Kabul and Silva (2014), “Machine learning is a branch of artificial intelligence that is concerned with building systems that require minimal human

intervention in order to learn from data and make accurate predictions". As Ki-Ngā-Kōpuku will evolve in future, machine learning and its application may significantly complement its development. For example, the addition of a new Cloud server or CIOS is currently thought to be a manual process. Machine learning techniques may automate this process. Another feature of Ki-Ngā-Kōpuku is in detecting exceptions and logging them. In reality, an exception may be an indicator of a system fault or a potential effort to launch an attack. Regardless of the reason, machine learning may help with studying such reasons and may also help to initiate honey-pot in such situations. This will help Ki-Ngā-Kōpuku to study new attacks and how they evolve, without giving the attacker any notion of awareness that the attack is detected. This, in turn, will help to fight cyber attacks and cyber crime more efficiently.

8.3.3 Cryptography

The security mechanism discussed in Chapter 6 illustrates how encryption is an integral part of some processing of the security mechanism. Further to that, Appendix B on page 211 lists the progress on further research on Ki-Ngā-Kōpuku where encryption plays a vital role both application integrity and data security for an application deployed using Ki-Ngā-Kōpuku. Thus, encryption is a notable technology that complements the security model. At the same time, this implies that Ki-Ngā-Kōpuku may complement any other security tool/technology and vice versa.

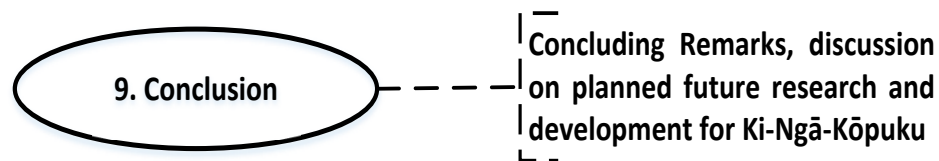
8.4 Conclusion

The research, as suggested in the above discussion, progressed according to the research design. The artefacts and deliverables are achieved as expected. Since Ki-Ngā-Kōpuku complements other technologies and security tools, it may strengthen other security approaches and vice versa. For computing and CC resilience, Ki-Ngā-Kōpuku can

be used as a stand-alone solution for the security aspects it is envisioned to provide, or it may be used in conjunction with other security tools or approaches. Also, there are a number of different computing technologies that may be integrated for further development and enhancement of Ki-Ngā-Kōpuku to add new features to it. The next chapter concludes this thesis, and possible future development of Ki-Ngā-Kōpuku is one of the aspects discussed.

Chapter 9

Conclusion



9.1 Introduction

The research presented is the initial research outcome. Ki-Ngā-Kōpuku is at its early childhood, and further research to enhance Ki-Ngā-Kōpuku is possible. This concluding explores the realisation on the research conducted in this section. In Section 9.2 on page 188, possible future and further research on Ki-Ngā-Kōpuku is outlined.

The research on Ki-Ngā-Kōpuku leads to a number of realisations. CC is gaining popularity and the rise in its popularity curve is apparent. CC offers benefits as well as challenges. The literature review reveals that different definitions exist for CC and thus there is no single standard definition. Some researchers include virtualisation and Internet as fulfilling criteria for CC. It may be argued that, while virtualisation is a significant technology behind the popularity of CC, it is a useful but not essential

defining criterion. This is based on the fact that any computing done by accessing remote resources is termed CC. Thus using a technology like virtualisation can be an accelerator for CC, but is not a pre-requisite for CC to exist. In the same way, it may be argued that the use of the word Internet may not be a universal defining criterion to define CC. Again, it means access to remote resources, which is currently done mostly via Internet. But with changes in technology, future access to remote resources might entail new concepts, new technologies, and new terms replacing Internet without affecting or forcing change to the definition of CC. Thus, it may be argued that the defining factor(s) for CC may not be confined to using contemporary technologies like virtualisation and the Internet. They are accelerators for CC, not defining factors.

CC is a conceptual approach to contemporary computing means. CC itself is a concept rather than being a technology. It encapsulates any other computing means. Thus the approach towards CC is rather a strategic approach to computing where remote computing resources under someone else's management are accessed to leverage the benefits of computing. The Cloud threat taxonomy developed and presented in Chapter 2 on page 26 indicates that the security concerns for CC come from all facets of computing and computer network-related settings. Recent developments of Cloud-related terminologies, for example, SaaS, PaaS or IaaS are merely levels of access to remote resources. These resources normally reside in a distributed manner in Cloud architectures. Thus, the distributed nature of CC shows that it is rather a variant of distributed computing. The major concerns of Cloud security as well as the existing opinion of the need for a distributed security model for CC mainly arise from the distributed nature and the distributed resource distribution in CC.

Another observation from the literature review is that, the current state of Cloud security is somewhat vague, or at least not well structured to adequately address Cloud security concerns. The current state of security does not satisfy the researchers, and security in the Cloud and its current level of sophistication have been marked as

‘confusing’ by some researchers. As well, as discussed in the literature review, opinions exist that the distributed nature of CC is an indicator of better suitability of distributed security models over non-distributed security models. Based on opinions that Cloud security is not up to the mark to date, ongoing research and new approaches towards Cloud security are sought.

The consensus among researchers that security is one of the major concerns in any kind of computing makes it imperative that security concerns for CC are probably greater than the sum of security concerns in all other computing approaches. It is such, because CC is an encapsulating wrapper for any kind of computing means. It may be argued that CC is not a technology to compete with other technologies, it is an strategic computing approach that facilitates all other computing means to suit various organisational need in a feasible way (e.g. cost reduction, simplification of IT infrastructure management, risk transfer, and so on).

In light of the above, the proposed security model, Ki-Ngā-Kōpuku provides security for computing settings like CC where the resources are distributed. In fact, Ki-Ngā-Kōpuku itself may be treated as a secure approach to application deployment in CC. The approach towards security offered by Ki-Ngā-Kōpuku is somewhat different to existing security solutions in that, it does not act as the ‘doorkeeper’. Instead, it empowers an application to be its own ‘doorkeeper’ through self-healing mechanism. On top of the concept of distribution, Ki-Ngā-Kōpuku embeds the aspect of decentralisation by ensuring that the system is not only distributed, but also distributed in such a way that any single distributed part does not contain the whole application resulting in the elimination of a single point of failure. In doing so, Ki-Ngā-Kōpuku adds redundancy as a feature which in turn shows the self-healing capability of the security model. The self-healing capability embedded with redundancy makes the security model capable of providing a truly distributed architecture with a self-defence mechanism – the kind of security model that is envisioned to be better suited to CC and security of the distributed

resources in the Cloud.

A security model that employs a security mechanism in a distributed and decentralised manner, needs to do so using a random approach. The randomisation of distribution is important as the main goal of distribution is to ensure that the entirety of an entity (i.e. data of application in this context) may not be visible or apparent to any unwanted parties. This is important so that an overall picture or map of the application cannot be imagined or constructed, which may in turn facilitate breaches or attacks. To achieve this, the elimination of a single point of failure comes as a condition. If single points of failure exists, the purpose of distribution may lose its appeal. Fulfilling the criteria of no single point of failure would demand decentralisation. A distributed security model like the proposed one also needs to ensure that there is no core of the system. Having any core of the system would leave a single point of failure, though there could be multiple single-points-of-failure existent in a scenario (e.g. backup or replica of system core). Ki-Ngā-Kōpuku addresses this by being decentralised. All the nodes in Ki-Ngā-Kōpuku have the same ‘power’ and thus there is no inferior or superior node in Ki-Ngā-Kōpuku – this makes it completely decentralised having no centralised system core.

In this concluding chapter, the planned future developments of Ki-Ngā-Kōpuku are also taken into account and discussed below.

9.2 Future Work

Research into Ki-Ngā-Kōpuku is ongoing. Some progress for future developments is already done and is presented in Appendix B on page 211 and Appendix C on page 232.

The proposed security model is in its first phase. It is expected that Ki-Ngā-Kōpuku will go through further improvements and enhancements as research progresses. The current level of performance expectation may be further refined by re-visiting the integrity of the existing design of the system. At the same time, new features will

be added to enhance the functionality of the proposed security model. At this stage, the following are some major aspects that would be taken into consideration in future research:

9.2.1 Architecture Dependency

It is argued that a security model consists of a RA and its underlying security model. It is not imperative that every security mechanism needs to have a custom RA. A security mechanism may work on existing architecture or conventional networking settings without the requirement for any tailored architectural arrangement like the one proposed in Chapter 5 on page 112.

In future research, the feasibility and options to make Ki-Ngā-Kōpuku architecture-independent will be explored. Referring back to the system architecture illustrated in Figure 5.2 on page 120, the distribution of nodes to the devices beyond BEC premise would be considered. If the regulations (e.g. governmental controls) are not violated, the distribution of nodes may occur in end-users' devices and the nodes may 'sleep' in those nodes. The nodes thus may use the computing power of the end-user devices when those devices are not in use. Such approach may lead to having the nodes of an application distributed to a massive number that would result in creating an unpredictable scenario to heighten security integrity. As a result, the probability of an application being compromised to make it unavailable may theoretically come down to almost zero.

9.2.2 Automation of Randomisation

Current research assumes that adding a new server and the random scenario at the BEC premise (where new CIOS may come alive at any time or an existing CIOS may go down) is a manual process. In future research, pre-built VMs as CIOS may be used to automate the process where the system will have another layer to control the

deliberate creation and deletion of the CIOs. The same applies to the process of adding new nodes. Automating the processes of adding or deleting CIOs and nodes would enable less human interaction in system management, and would make Ki-Ngā-Kōpuku suitable for applications that may be deployed in remote places or contexts with difficult or hazardous access. Example of such systems may be under-sea monitoring systems, systems used in space travel-related apparatus, systems used in battlefields, and so on.

9.2.3 Data Security

The proposed model, provides application security, context illiteracy and self-healing features, as discussed in Section 4.4 on page 109. Providing data security is a planned enhancement for Ki-Ngā-Kōpuku. In this regard, some research is carried out and listed in Appendix B on page 211 and Appendix C on page 232 where the initial research shows a modified version of Ki-Ngā-Kōpuku that provides application security as well as data security. However, the research presented in Appendix B on page 211 and Appendix C on page 232 is at a very early stage and may be associated with integrity issues. This is part of future planned enhancement that will be further developed and fully tested as research progresses.

9.2.4 Incident Monitoring

The current proposed model is self healing and does not explicitly need to deal with situations when an exception is generated. Currently, as response to an exception, it may simply discard the affected node and continue with the redundant copies of a node. In future developments of Ki-Ngā-Kōpuku, the exceptions and the root cause of the exceptions will be examined by the system. As discussed previously, Ki-Ngā-Kōpuku may be enhanced by using machine learning approaches. The exception may be analysed as to whether to create a honey-pot to study an ongoing attack without the

attacker being aware, or the exception may result in force closing down of a node or a server. Thus, incident monitoring and response will be further enhanced in future developments of the security model.

9.2.5 Existing Performance Bottleneck

The birth of Ki-Ngā-Kōpuku is marked by the research presented and thus it would make sense to think of the maturity of Ki-Ngā-Kōpuku as at a very early childhood level. It comes with performance bottlenecks. Given today's computing power, performance bottlenecks may be experienced, though this may not be the case with tomorrow's computing capability, as the computing capability of the devices is increasing exponentially. Currently, all the nodes may take part in collaborative processing (e.g. component distribution). If the number of participating nodes in a process could be minimized, the performance of Ki-Ngā-Kōpuku would be improved. Minimising or limiting the number of participating nodes is thus one of the crucial future modifications envisioned for Ki-Ngā-Kōpuku.

9.3 Concluding Remarks

Ki-Ngā-Kōpuku is a security model in the form of an application development architecture. It proposes a novel approach towards security. Applications developed and deployed using the Ki-Ngā-Kōpuku approach would come with the capability of self-healing, would be distributed and decentralised with no system core and, as a result, with no single point of failure. High level of randomization and redundancy of Ki-Ngā-Kōpuku makes its architectural context confusing to the outside world and a mapping of the architecture to completely shut down an application deployed using Ki-Ngā-Kōpuku may not be possible for an attacker.

The future envisioned development for Ki-Ngā-Kōpuku requires evaluation of

computing technologies. In future when Ki-Ngā-Kōpuku considers data security, off-the-shelf OS or computing platforms may not be suitable for the security model. Custom-made CIOS built specifically for Ki-Ngā-Kōpuku may become a requirement as the security model continues to evolve. For example, an Object Oriented Programming (OOP) language - if used to deploy an application using Ki-Ngā-Kōpuku – is capable of developing and deploying an application using the security model, but the deployment may not be as optimised as it is envisioned as the development would be constrained by the features of the OOP tool(s) used.

The full potential of Ki-Ngā-Kōpuku is yet to be explored. Ki-Ngā-Kōpuku can be used to develop and deploy an application regardless of the type of application, data used by the application, tools used to develop the application, and the purpose and context of the application. The current state of Ki-Ngā-Kōpuku, as suggested earlier, indicates that it may not be suitable for some real-time systems, but would be as a result of its future development. It is up to developers to decide whether using Ki-Ngā-Kōpuku would be cost-effective and would meet the performance threshold for the application of concern.

Ki-Ngā-Kōpuku is a big project that incorporates diverse ideas, approaches and technologies. The proposed framework has a long way to go. If nurtured properly with gradual enhancement, Ki-Ngā-Kōpuku will become a nightmare for the cyber criminals.

References

- Agrawal, R. (1993). Mining association rules between sets of items in large databases..
- Ahmed, M., Litchfield, A. & Ahmed, S. (2014, December). A generalized threat taxonomy for cloud computing. In *25th australasian conference on information systems*. Auckland, New Zealand: ACIS 2014.
- Ahmed, M., Litchfield, A. & Sharma, C. (2016). A distributed security model for cloud computing. In *Proceedings of the americas conference on information systems (AMCIS)*. San Diego, USA.
- Ahmed, M. & Litchfield, A. T. (2016, September). Taxonomy for identification of security issues in cloud computing environments. *Journal of Computer Information Systems*. (DOI: 10.1080/08874417.2016.1192520)
- Akande, A., April, N. & Belle, J. (2013, December). Management issues with cloud computing. In *ACM ICC3'13* (p. 119-124). Wuhan, China: ACM.
- Alfath, A., Baina, K. & Baina, S. (2013, April). Cloud computing security: Fine-grained analysis and security approaches. In *IEEE 2013 national security days (JNS3) conference* (p. 1-6). Rabat: IEEE. doi: 10.1109/JNS3.2013.6595465
- Alva, A., Caleff, O., Elkins, G., Lum, A., Pasley, K., Sudarsan, S., ... Venkitaraman, R. (2013). The notorious nine cloud computing top threats in 2013. *Cloud Security Alliance, February*, 1-21.
- AlZain, M. A., Soh, B. & Pardede, E. (2012). A new model to ensure security in cloud computing services. *Journal of Service Science Research*, 4, 49-70. doi: 10.1007/s12927-012-0002-5
- Ameller, D., Ayala, C., Cabot, J. & Franch, X. (2012, September). How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE international requirements engineering conference (RE)*. IEEE.
- Antoni, M. & Ammad, N. (2011). Practical formal validation method for interlocking systems. In *Practical formal validation method for interlocking systems* (p. 1-15).
- Apple. (2014, 24September). *icloud: What is icloud* [Internet web page]. Retrieved from <http://support.apple.com/kb/ph2608>
- Arazy, O., Kumar, N. & Shapira, B. (2010). A theory-driven design framework for social recommender systems. *Journal of the Association for Information Systems (JAIS)*, 11(9), 455-490.
- Arlitsch, K. & Edelman, A. (2014). Staying safe: Cyber security for people and organizations. *Journal of Library Administration*, 54(1), 46-56.
- Arshad, J., Townend, P. & Xu, J. (2013). A novel intrusion severity analysis approach

- for clouds. *Future Generation Computer Systems*, 29, 416–428.
- Azeemi, I. K., Lewis, M. & Tryfonasc, T. (2013, March 19-22). Migrating to the cloud: Lessons and limitations of ‘traditional’ IS success models. In *Conference on systems engineering research (CSER’13)* (Vol. 16, pp. 737–746). Atlanta, GA, USA.
- Bakry, S. H. & Alfantookh, A. (2006). It-governance practices: COBIT. Saudi Computer Society.
- Balliu, M. & Mastroeni, I. (2009). A weakest precondition approach to active attacks analysis. In *ACM PLAS ’09* (p. 59-71). ACM.
- BBC. (2017). *Password manager onelogin hit by data breach* [Internet web page]. Retrieved from <http://www.bbc.com/news/technology-40118699>
- Beckers, K., Côté, I. & Goeke, L. (2014). A catalog of security requirements patterns for the domain of cloud computing systems. In *ACM SAC’14* (p. 337-342). Seoul, Korea. doi: <http://dx.doi.org/10.1145/2554850.2554921>
- Behl, A. & Behl, K. (2012). An analysis of cloud computing security issues. In *World congress on information and communication technologies (WICT)* (p. 109-114). Trivandrum, India.: IEEE. doi: 10.1109/WICT.2012.6409059
- Benbasat, I. & Zmud, R. (1999). Empirical research in information systems: The practice of relevance. *MIS Quarterly*, 23(1), 3-16.
- Berger, S., Caceres, R., Goldman, K., Perez, R., Sailer, R. & van Doorn, L. (2006). vtpm: virtualizing the trust platform module. In *USENIX-SS’06: Proceedings of the 15th conference on unix security symposium*. Berkeley, CA, USA.
- Bhagyavati & Hicks, G. (2003, October). A basic security plan for a generic organization. In *CCSC: Rocky mountain conference* (Vol. 19, p. 248-256).
- Bloomberg. (2011a). *Amazon.com server said to have been used in sony attack* [Internet web page]. Retrieved from <http://www.bloomberg.com/news/2011-05-13/sony-network-said-to-have-been-invaded-by-hackers-using-amazon-com-server.html>
- Bloomberg. (2011b). *Sony network breach shows amazon cloud’s appeal for hackers* [Internet web page]. Retrieved from <http://www.bloomberg.com/news/2011-05-15/sony-attack-shows-amazon-s-cloud-service-lures-hackers-at-pennies-an-hour.html>
- Bottino, L. J. & Hughes, W. J. (2006, October). Security measures in a secure computer communications architecture. In *25th digital avionics systems conference* (p. 6B3.1-6B3.18).
- Bouayad, A., Blilat, A., el houda Mejhed, N. & Ghazi, M. E. (2012). Cloud computing: Security challenges. In *Colloquium in information science and technology (CIST)* (p. 26-31). Fez: IEEE. doi: 10.1109/CIST.2012.6388058
- Bowen, G. A. (2006, September). Grounded theory and sensitizing concepts. *International Journal of Qualitative Methods*, 5(3).
- Bowles, M. (2012). The business of hacking and birth of an industry. *Bell Labs Technical Journal*, 17(3), 5-16.
- Bradley, T. (2015). *Zappos hacked: What you need to know* [Internet web page]. Retrieved from <http://www.pcworld.com/article/248244/zappos>

- `_hacked_what_you_need_to_know.html`
- Buyyaa, R., Yea, C. S., Venugopala, S., Broberg, J. & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 599–616.
- Cardoso, A. & Simões, P. (2012). Cloud computing: Concepts, technologies and challenges. In *Vinorg 2011, CCIS 248* (p. 127–136). Springer-Verlag Berlin Heidelberg.
- Casalicchio, E. & Silvestri, L. (2013). Mechanisms for SLA provisioning in cloud-based service providers. *Computer Networks*, 57, 795–810.
- Cascella, R. G., Morin, C., Harsh, P. & Jegou, Y. (2012). Contrail: A reliable and trustworthy cloud platform. In *ACM EWDCC '12*. ACM.
- Casola, V., Cuomo, A., Rak, M. & Villano, U. (2013). The cloudgrid approach: Security analysis and performance evaluation. *Future Generation Computer Systems*, 29, 387–401.
- Cesay, E. N., Chandrasekaran, C. & Simpson, W. R. (2010). An authentication model for delegation, attribution and least privilege. In *ACM PETRA '10*. ACM.
- Celesti, A., Fazio, M., Villari, M. & Puliafito, A. (2012). Virtual machine provisioning through satellite communications in federated cloud environments. *Future Generation Computer Systems*, 28, 85–93.
- CERT. (2017). *Wannacry ransomware used in large scale international attacks* [Internet web page]. Retrieved from <https://www.cert.govt.nz/businesses-and-individuals/recent-threats/alert-wannacry-ransomware-used-in-large-scale-international-attacks/>
- Chadwick, D. W. & Fatema, K. (2012). A privacy preserving authorisation system for the cloud. *Journal of Computer and System Sciences*, 78, 1359–1373.
- Chamberlain, B. (2009). Phenomenology: a qualitative method. *Clinical nurse specialist CNS*, 23 2, 52-3.
- Che, J., Duan, Y., Zhang, T. & Fan, J. (2011). Study on the security models and strategies of cloud computing. In *International conference on power electronics and engineering application* (pp. 586–593).
- Cheng, F. (2011). Security attack safe mobile and cloud-based one-time password tokens using rubbing encryption algorithm. *Mobile Networks Appl*, 16, 304–336.
- Chou, T. (2013). Security threats on cloud computing vulnerabilities. *Int J Comput Sci Inf Technol*, 5(3), 79-88.
- Chow, R., Jakobsson, M., Masuoka, R., Molina, J., Niu, Y., Shi, E. & Song, Z. (2010, October). Authentication in the clouds: A framework and its application to mobile users. In *ACM CCSW'10* (p. 1-6). Chicago, Illinois, USA: ACM.
- Chow, R. & Jonas, W. (2008). Beyond dualism in methodology: An integrative design research medium "maps" and some reflections. In *Proceedings of the 2008 design research society conference*. Sheffield.
- CNN. (2011). *Dropbox's password nightmare highlights cloud risks*. [Internet web page]. Retrieved from http://money.cnn.com/2011/06/22/technology/dropbox_{}_passwords/

- CNN. (2014). *JPMorgan: 76 million customers hacked*. [Internet web page]. Retrieved from <http://money.cnn.com/2014/10/02/technology/security/jpmorgan-hack/>
- Computerworld. (2014). *Jpmorgan chase breach affected 83 million customers*. [Internet web page]. Retrieved from <http://www.computerworld.co.nz/article/556581/jpmorgan-chase-breach-affected-83-million-customers>
- Corbin, J. M. (1990). Grounded theory research : Procedures , canons , and evaluative criteria. *Qualitative Sociology*, 13(1), 3-21.
- Creswell, J. W. (2003). *Design : Qualitative , quantitative , and mixed method approaches*. SAGE Publications.
- Dahbur, K., Mohammad, B. & Tarakji, A. B. (2011, April). A survey of risks, threats and vulnerabilities in cloud computing. In *ACM ISWSA'11*. Amman, Jordan: ACM.
- Dai, H., Zhao, S., Zhang, J., Qiu, M. & Tao, L. (2015). Security enhancement of cloud servers with a redundancy-based fault-tolerant cache structure. *Future Generation Computer Systems*, 52, 147-155.
- Deibert, R. (2012). *Distributed security as cyber strategy: Outlining a comprehensive approach for canada in cyberspace* (Research Paper). The Citizen Lab and Canada Centre for Global Security Studies, Munk School of Global Affairs, University of Toronto.
- Demont, C., Breitenbücher, U., Kopp, O., Leymann, F. & Wettinger, J. (2013). Towards integrating TOSCA and ITIL. In *ZEUS*. Retrieved from <http://ceur-ws.org/Vol-1029/paper6.pdf>
- Disterer, G. (2013). ISO / IEC 27000 , 27001 and 27002 for information security management. *Journal of Information Security*, 4, 92-100.
- Dixon-Woods, M., Agarwal, S., Jones, D., Young, B. & Sutton, A. (2005). Synthesising qualitative and quantitative evidence: a review of possible methods. *Journal of health services research & policy*, 10 1, 45-53.
- Dukaric, R. & Juric, M. B. (2013). Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems*, 29, 1196–1210.
- Duncan, A., Creese, S. & Goldsmith, M. (2012). Insider attacks in cloud computing. In *IEEE 11th international conference on trust, security and privacy in computing and communications* (p. 857–862). IEEE.
- Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y. & Hamilton, D. (1998). Experiences using lightweight formal methods for requirements modelling. *IEEE Transactions on Software Engineering*, 24(1).
- Emekaroha, V., Netto, M., Calheiros, R., Brandic, I., Buyya, R. & Rose, C. (2012). Towards autonomic detection of sla violations in cloud infrastructures. *Future Gener Comput Syst*, 28, 1017–1029.
- Endo, P. T., Rodrigues, M., Goncalves, G. E., Kelner, J., Sadok, D. H. & Curescu, C. (2016). High availability in clouds: systematic review and research challenges. *Journal of Cloud Computing: Advances, Systems and Applications*, 5, 1-15.
- Enterprise information systems in 21st century: Opportunities and challenges*. (2009).

- Deep and Deep Publications.
- Esayas, S. Y. (2012). A walk in to the cloud and cloudy it remains: The challenges and prospects of 'processing' and 'transferring' personal data. *Computer Law & Security Review*, 28, 662–678.
- Eslahi, M., Naseri, M. V., Hashim, H., Tahir, N. M. & Saad, E. H. M. (2014). Byod: Current state and security challenges. In *IEEE symposium on computer applications & industrial electronics iscaie* (p. 189-192). Penang, Malaysia: IEEE.
- Fan, C., S & Huang. (2013). Controllable privacy preserving search based on symmetric predicate encryption in cloud storage. *Future Gener Comput Syst*, 29, 1716-1724.
- Fernandes, D. A. B., Soares, L. F. B., Gomes, J. V., Freire, M. M. & Inacio, P. R. M. (2014). Security issues in cloud environments: a survey. *International Journal of Information Security*, 13, 113-170. doi: 10.1007/s10207-013-0208-7
- Fernandez, E. B. & Monge, R. (2014). A security reference architecture for cloud systems. In *ACM ICSA '14*. Sidney, NSW, Australia: ACM.
- Fernando, N., Loke, S. W. & Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29, 84–106.
- Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B. & Zhang, X. (2006). Virtual clusters for grid communities. In *CCGRID, IEEE computer society* (p. 513-520). IEEE.
- FoxNewsTech. (2017). *Military personnel data leaked in dun & bradstreet database* [Internet web page]. Retrieved from <http://www.foxnews.com/tech/2017/03/16/military-personnel-data-leaked-in-dun-bradstreet-database.html>
- Franceschi-Bicchierai, L. (2014, November). *What we know about 'regin' spy malware* [Internet web page]. Retrieved from <http://www.stuff.co.nz/technology/digital-living/63550458/what-we-know-about-regin-spy-malware.html>
- Garcia-Morchon, O., Kuptsov, D., Gurtov, A. & Wehrle, K. (2013, July). Cooperative security in distributed networks. *Computer Communications*, 36(12), 1284-1297. doi: 10.1016/j.comcom.2013.04.007
- Ghebghoub, Y., Boussaid, O. & Oukid, S. (2014). Security model based encryption to protect data. In *ACM ISDOC* (p. 50-55). ACM. doi: <http://dx.doi.org/10.1145/2618168.2618176>
- Gill, A., Chew, E., Bird, G. & Kricker, D. (2015). An agile service resilience architecture capability: Financial services case study. *2015 IEEE 17th Conference on Business Informatics*, 1, 209-216.
- Gill, A., Chew, E., Kricker, D. & Bird, G. (2016). Adaptive enterprise resilience management: Adaptive action design research in financial services case study. *2016 IEEE 18th Conference on Business Informatics (CBI)*, 01, 113-122.
- Gill, A. Q., Phennel, N., Lane, D. & Phung, V. L. (2016, June). Iot-enabled emergency information supply chain architecture for elderly people: The australian context. *Information Systems*, 58.
- Gollmann, D. (2010). Computer security. *WIREs Computational Statistics*, 2, 544-554.
- Gonzalez, N., Miers, C., Redigolo, F., Simplicio, M., Carvalho, T., Naslund, M. &

- Pourzandi, M. (2012). A quantitative analysis of current security concerns and solutions for cloud computing. *Journal of Cloud Computing*, 1(11), 1–18.
- Gregor, S. & Jones, D. (2007). The anatomy of a design theory. *Journal of the Association for Information Systems (JAIS)*, 8(5), 312–335.
- Gritzalis, S., Mitchell, C., Thuraisingham, B. & Zhou, J. (2014a). Security in cloud computing. *International Journal of Information Security*, 13, 95–96.
- Gritzalis, S., Mitchell, C., Thuraisingham, B. & Zhou, J. (2014b). Security in cloud computing. *International Journal of Information Security*, 13, 95–96. doi: 10.1007/s10207-014-0232-2
- Grobauer, B., Walloschek, T. & Stocker, E. (2011). Understanding cloud computing vulnerabilities. *Cloud Computing, March/April 2011*, 50–57.
- Gross, A. & Doerr, J. (2012, September). What do software architects expect from requirements specifications? In *2012 IEEE first international workshop on the twin peaks of requirements and architecture (twin peaks)*. Chicago, Illinois, USA: IEEE.
- Guo, Q., Sun, D., Chang, G., Sun, L. & Wang, X. (2011). Modeling and evaluation of trust in cloud computing environments. In *3rd international conference on advanced computer control (icacc 2011)* (p. 112–116).
- Hall, P., Dean, J., Kabul, I. K. & Silva, J. (2014). *An overview of machine learning with sas @enterprise miner™* [Internet web page]. Retrieved from <https://support.sas.com/resources/papers/proceedings14/SAS313-2014.pdf>
- Han, J., Susilo, W. & Mu, Y. (2013). Identity-based data storage in cloud computing. *Future Generation Computer Systems*, 29, 673–681.
- Haniff, D. & Baber, C. (1999). Wearable computers for the fire service and police force: Technological and human factors. In *ISWC '99 proceedings of the 3rd IEEE international symposium on wearable computers* (p. 185–186).
- Hashemi, S. & Ardakani, M. (2012). Taxonomy of the security aspects of cloud computing systems – a survey. *Int J Appl Inf Syst*, 4(1), 21–28.
- Hawkey, K., Gagne, A., Botta, D., Beznosov, K., Werlinger, R. & K, K. M. (2008). Human, organizational and technological factors of it security. In *CHI 2008 proceedings, florance, italy* (p. 3639–3644).
- Hein, D., Morozov, S. & Saiedian, H. (2012). A survey of client-side web threats and counterthreat measures. *Security and Communication Networks*, 5, 535–544. doi: 10.1002/sec.349
- Hernandez-Ramirez, E. M., Sosa-Sosa, V. J. & Lopez-Arevalo, I. (2012). A comparison of redundancy techniques for private and hybrid cloud storage. *Journal of Applied Research and Technology*, 10, 893–901.
- Hevner, A., March, S., Park, J. & Ram, S. (2004). Design science in is research. *MIS Quarterly*, 28(1), 75–106.
- IdentityForce. (2017). *2017 data breaches - the worst so far* [Internet web page]. Retrieved from www.identityforce.com/blog/2017-data-breaches
- Jaeger, P., Lin, J. & Grimes, J. (2008). Cloud computing and information policy:

- Computing in a policy cloud? *Journal of Information Technology & Politics*, 5(3), 269-283. doi: 10.1080/19331680802425479
- Jorissen, K., Vila, F. & Rehr, J. (2012). A high performance scientific cloud computing environment for materials simulations. *Computer Physics Communications*, 183, 1911–1919.
- Joshi, G., Soljanin, E. & Wornell, G. (2015). Efficient redundancy techniques for latency reduction in cloud systems. In *Allerton conference on communication, control and computing*.
- Journal, T. W. S. (2014, 25September). *Apple denies icloud breach* [Internet web page]. Retrieved from <http://online.wsj.com/articles/apple-celebrity-accounts-compromised-by-very-targeted-attack-1409683803>
- Kaiya, H., Sato, T., Osada, A., Kitazawa, N. & Kaijiri, K. (2008, March). Toward quality requirements analysis based on domain specific quality spectrum. In *ACM SAC'08* (p. 596-601). Fortaleza, Cear'a, Brazil: ACM.
- Keeling, M. (2014, October). *Reflections on software engineering* [Internet web page]. Retrieved from <http://www.neverletdown.net/2014/10/dealing-with-constraints-in-software-architecture.html>
- Khalil, I. M., Khreishah, A., Bouktif, S. & Ahmad, A. (2013). Security concerns in cloud computing. In *10th international conference on information technology: New generations* (p. 411-416). IEEE. doi: 10.1109/ITNG.2013.127
- Khamadja, S., Adi, K. & Logrippo, L. (2013). Designing flexible access control models for the cloud. In *ACM SIN'13* (p. 225-232). Aksaray, Turkey: ACM. doi: <http://dx.doi.org/10.1145/2523514.2527005>
- Khan, A. N., Kiah, M. M., Khan, S. U. & Madani, S. A. (2013). Towards secure mobile cloud computing: A survey. *Future Generation Computer Systems*, 29, 1278–1299.
- Khan, K. & Malluhi, Q. (2010, September/October). *Establishing trust in cloud computing*. IT Pro.
- Khan, S. N. (2014). Qualitative research method-phenomenology. *Asian Social Science*, 10(21), 298-310.
- Khorshed, M. T., Ali, A. S. & Wasimi, S. A. (2012). A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems*, 28, 833–851.
- King, N. J. & Raja, V. T. (2012). Protecting the privacy and security of sensitive customer data in the cloud. *Computer Law and Security Review*, 8, 308-319.
- Kizza, J. M. (2009). *Computer communications and networks*. Springer-Verlag London Limited. doi: 10.1007/978-1-84800-917-2_{ }3
- Krombholz, K., Hobel, H., Huber, M. & Weippl, E. (2013, November). Social engineering attacks on the knowledge worker. In *ACM SIN '13* (p. 28–35). Aksaray, Turkey: ACM.
- Kueppers, S. & Schillingno, M. (1999). Getting our act together: Human and technological factors in establishing an on-line knowledge base. In *SIGUCCS 99* (p. 135–139). Denver, Colorado, USA: ACM.

- Kuhn, R., Chandramouli, R. & Butler, R. W. (2002). *Cost effective use of formal methods in verification and validation*. Retrieved from <https://www.nist.gov/publications/cost-effective-use-formal-methods-verification-and-validation-foundations>
- Kulkarni, G., Gambhir, J., Patil, T. & Dongare, A. (2012). A security aspects in cloud computing. In *IEEE 3rd international conference on software engineering and service science (ICSESS)* (p. 547-550). Beijing, China: IEEE. doi: 10.1109/ICSESS.2012.6269525
- Kulkarni, P. & Khanai, R. (2015). Addressing mobile cloud computing security issues: a survey. In *IEEE ICCSP 2015 conference* (p. 1463–1467). Bangalore, India.
- Kumar, P., Nitin, Sehgal, V., Shah, K. & Chauhan, D. S. (2011). A novel approach for security in cloud computing using hidden markov model and clustering. In *2011 world congress on information and communication technologies (wict)*. IEEE.
- Kushner, D. (2013, February). *The real story of stuxnet* [Internet web page]. Retrieved from <http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet>
- Lauesen, S. (2002). *Software requirements: styles and techniques*. Pearson Education Limited. Great Britain.
- Liang, C. & Yu, F. R. (2015). Wireless network virtualization: A survey, some research issues and challenges. *IEEE Communication Survays & Tutorials*, 17(1), 358-380. doi: 10.1109/COMST.2014.2352118
- Lin, A. & Chen, N. (2012). Cloud computing as an innovation: Perception, attitude and adoption. *International Journal of Information Management*, 32, 533-540.
- Lin, X. (2014). A multi-redundancy structure model of cloud computing digital library. *Applied Mechanics and Materials*, 519-520, 137-141.
- Liu, W. (2012). Research on cloud computing security problem and strategy. In *2nd international conference on consumer electronics, communications and networks (cecnet)* (p. 1216-1219). Yichang: IEEE. doi: 10.1109/CECNet.2012.6202020
- Lo, C., Huang, C. & Ku, J. (2008). Cooperative intrusion detection system framework for cloud computing networks. In *First IEEE international conference on ubi-media computing* (p. 280-284). IEEE.
- Lombardi, F. & Pietro, R. D. (2011). Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34, 1113–1122.
- Mackay, M., Baker, T. & Al-Yasiri, A. (2012). Security-oriented cloud computing platform for critical infrastructures. *Computer Law & Security Review*, 28, 679–686.
- Malan, R. & Bredemeyer, D. (2001). *Functional requirements and use cases* [Internet web page]. Retrieved from http://www.bredemeyer.com/pdf_files/functreq.pdf
- March, S. T. & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4), 251-266.
- Markus, M., Majchrzak, A. & Gasser, L. (2002). A design theory for systems that support emergent knowledge processes. *MIS Quarterly*, 26(3), 179-212.
- Mingers, J. (2001). Combining is research methods: Towards a pluralist methodology.

- Information Systems Research*, 12, 240-259.
- Modi, C., Patel, D., Borisaniya, B., Patel, A. & Rajarajan, M. (2013). A survey on security issues and solutions at different layers of cloud computing. *Journal of Supercomputing*, 63, 561-592. doi: 10.1007/s11227-012-0831-5
- Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A. & Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36, 42–57.
- Mohamadi, M. & Ranjbaran, T. (2013). Effective factors on the success or failure of the online payment systems, focusing on human factors. In *7th international conference on e-commerce in developing countries with focus of e-security*. Iran: IEEE.
- Mohamed, E. M. & Abdelkader, H. S. (2012, May). Enhanced data security model for cloud computing. In *The 8th international conference on informatics and systems (INFOS2012)* (p. cc12-cc17).
- Newman, R. C. (2006, September). Cybercrime, identity theft, and fraud: Practicing safe internet - network security threats and vulnerabilities. In *ACM infoseccd conference '06* (p. 68-78). Kennesaw, GA, USA.
- Noor, T., Sheng, Q., Zeadally, S. & Yu, J. (2013). Trust management of services in cloud environments: obstacles and solutions. *ACM Comput Surv*, 46(1), 12:1–12:30.
- Nunamaker, J., Chen, M. & Purdin, T. (1991). System development in information systems research. *Journal of Management Information Systems*, 7(3), 89-106.
- NZHerald. (2014). *Global wannacry cyber attack could be devastating* [Internet web page]. Retrieved from http://www.nzherald.co.nz/business/news/article.cfm?c_id=3&objectid=11856377
- Offermann, L., Levina, O., Sch onherr, M. & Bub, U. (2009). Outline of a design science research process. In *ACM DESRIST'09*.
- Onwubiko, C. & Lenaghan, A. P. (2007). Managing security threats and vulnerabilities for small to medium enterprises. *IEEE Intelligence and Security Informatics*, 244-249.
- Osborne, C. (2017). *Intercontinental hotels group admits data breach* [Internet web page]. Retrieved from <http://www.zdnet.com/article/intercontinental-hotels-group-admits-data-breach/>
- Paech, B., Dutoit, A. H., Kerkow, D. & von Knethen, A. (2002). Functional requirements, non-functional requirements, and architecture should not be separated –a position paper existing solutions..
- Pal, S., Mandal, A. K. & Sarkar, A. (2015). Application multi-tenancy for software as a service. *ACM SIGSOFT Software Engineering Notes*, 40(2), 1-8. doi: 10.1145/2735399.2735412
- Pamies-Juarez, L., Garcia-Lopez, P., Sanchez-Artigas, M. & Herrera, B. (2011). Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures. *Computer Networks*, 55, 1100-1113.
- Pandit, H. J., O'Sullivan, D. & Lewis, D. (2018). GDPR data interoperability model.. Retrieved from <https://pdfs.semanticscholar.org/3a41/a4a3cf15842874c2aa42f5893cc311e2d8d5.pdf>

- Park, J. H. (2012). A virtual security framework for public cloud computing. In *Computer science and its applications, lecture notes in electrical engineering* (p. 421-428). Springer Science Business Media Dordrecht.
- Patel, A., Taghavi, M., Bakhtiyari, K. & Junior, J. C. (2013). An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36, 25–41.
- Patel, H. B., Patel, D. R., Borisaniya, B. & Patel, A. (2012). Data storage security model for cloud computing. In V. Das & J. Stephen (Eds.), *CNC 2012, LNICST* (p. 37-45). Institute for Computer Sciences, Social Informatics and Telecommunications Engineering.
- Paul, I. (2015, 16 June). *The lastpass security breach: what you need to know, do, and watch out for* [Internet web page]. Retrieved from <http://www.pcworld.com/article/2936621/the-lastpass-security-breach-what-you-need-to-know-do-and-watch-out-for.html>
- PCWorld. (2014, 24September). *Don't blame icloud yet for hacked celebrity nudes* [Internet web page]. Retrieved from <http://www.pcworld.com/article/2601081/dont-blame-icloud-yet-for-hacked-celebrity-nudes.html>
- Pearce, M., Zeadally, S. & Hunt, R. (2013). Virtualization: Issues, security threats, and solutions. *ACM Computing Surveys*, 45(2), 17:1-17:39.
- Perez-Botero, D., Szefer, J. & Lee, R. (2013, May). Characterizing hypervisor vulnerabilities in cloud computing servers. In *Cloudcomputing'13* (p. 3-10). Hangzhou, China: ACM.
- Petcu, D., Macariu, G., Panica, S. & Craciun, C. (2013). Portable cloud application – from theory to practice. *Future Generation Computer Systems*, 29, 1417–1430.
- Poh, G. S., Nazir, M. A. N. M., Goi, B.-M., Tan, S.-Y., Phan, R. C.-W. & Shamsudin, M. S. (2013). An authentication framework for peer-to-peer cloud. In *ACM SIN '13* (p. 94-101). Aksaray, Turkey: ACM. doi: <http://dx.doi.org/10.1145/2523514.2523531>
- Pree, W. (1995). *Design patterns for object-oriented software development*. Addison-Wesley.
- Pries-Heje, J., Baskerville, R. & Venable, J. R. (2008). Strategies for design science research evaluation. In *European conference on information systems*. AIS.
- Purao, S. & Storey, V. C. (2008). Evaluating the adoption potential of design science efforts: The case of apsara. *Decision Support Systems*, 44, 369-381.
- Rabai, L. B. A., Jouini, M., Aiss, A. B. & Mili, A. (2013). A cybersecurity model in cloud computing environments. *Journal of King Saud University — Computer and Information Sciences*, 25, 63–75.
- Ragan, S. (2017). *Esea hacked, 1.5 million records leaked after alleged failed extortion attempt* [Internet web page]. Retrieved from <https://www.csoononline.com/article/3155397/security/esea-hacked-1-5-million-records-leaked-after-alleged-failed-extortion-attempt.html>
- Randles, M., Lamb, D., Odat, E. & Taleb-Bendiab, A. (2011). Distributed redundancy

- and robustness in complex systems. *Journal of Computer and System Sciences*, 77, 293-304.
- Raza, S., Duquennoy, S., Hoglund, J., Roedig, U. & Voigt, T. (2012). Secure communication for the internet of things — a comparison of link-layer security and ipsec for 6lowpan. *Security and Communication Networks, Special Issue*, 1-15. doi: 10.1002/sec.406
- Roberts, J. C. & Al-Hamdani, W. (2011). Who can you trust in the cloud? a review of security issues within cloud computing. In *Information security curriculum development conference*. ACM.
- Robling, G. & Muller, M. (2009, July). Social engineering: A serious underestimated problem. In *ACM iticse'09* (p. 384–387). Paris, France: ACM.
- Rong, C., Nguyen, S. T. & Jaatun, M. G. (2013). Beyond lightning: A survey on security challenges in cloud computing. *Computers and Electrical Engineering*, 39, 47–54.
- Rosemann, M. & Vassey, I. (2008). Towards improving the relevance of information systems research to practice: The role of applicability checks. *MIS Quarterly*, 32(1), 1-22.
- Ryan, P. & Falvey, S. (2012). Trust in the clouds. *Computer Law & Security Review*, 28, 513–521.
- Sabahi, F. (2011). Cloud computing security threats and responses. In *IEEE 3rd international conference on communication software and networks (ICCSN)* (p. 245-249). Xi'an: IEEE. doi: 10.1109/ICCSN.2011.6014715
- Schwartz, M. (2015). *Zappos breach: 8 lessons learned* [Internet web page]. Retrieved from <http://www.darkreading.com/attacks-and-breaches/zappos-breach-8-lessons-learned/d/d-id/1102303?>
- Sevres, N. K. D. & Kakavand, H. (2017). *The blockchain revolution: An analysis of regulation and technology related to distributed ledger technologies* [Internet web page]. Retrieved from www.dlapiper.com
- Shaikh, F. B. & Haider, S. (2011, December). Security threats in cloud computing. In *6th international conference on internet technology and secured transactions (icitst)* (p. 214-219). UAE: IEEE.
- Singh, A. & Shrivastava, M. (2012). Overview of attacks on cloud computing. *Int J Eng Innovative Technol*, 1(4), 321–323.
- Soares, L., Fernandes, D., Gomes, J., Freire, M. & Inácio, P. (2014). Cloud security: State of the art. *Security, Privacy and Trust in Cloud Systems*, 3.
- Srinivasan, M. K., Sarukesi, K., Rodrigues, P., Manoj, S. & Revathy. (2012). State-of-the-art cloud computing security taxonomies – a classification of security challenges in the present cloud computing environment. In *ICACCI '12 proceedings of the international conference on advances in computing, communications and informatics* (p. 470-476). ACM. doi: 10.1145/2345396.2345474
- Srivastava, P., Singh, S., Pinto, A. A., Verma, S., Chaurasiya, V. K. & Gupta, R. (2011). An architecture based on proactive model for security in cloud computing. In *IEEE - international conference on recent trends in information technology* (p. 661-666). MIT, Anna University, Chennai, India.

- Starks, H. & Trinidad, S. B. (2007). Choose your method: a comparison of phenomenology, discourse analysis, and grounded theory. *Qualitative health research*, 17 10, 1372-80.
- Stuff.co.nz. (2014). *Spark broadband still down for many*. [Internet web page]. Retrieved from <http://www.stuff.co.nz/business/10468128/Spark-broadband-still-down-for-many>
- Sumter, L. (2010). Cloud computing: Security risk. In *ACM SE 48th annual southeast regional conference*. NY, USA: ACM. doi: 10.1145/1900008.1900152
- Sun, L., Singh, J. & Hussain, O. K. (2012, December). Service level agreement (sla) assurance for cloud services: A survey from a transactional risk perspective. In *ACM momm2012* (p. 263-266). Bali, Indonesia: ACM.
- Svantesson, D. & Clarke, R. (2010). Privacy and consumer risks in cloud computing. *Computer Law and Security Review*, 26, 391-397.
- Symantec. (2017). *What you need to know about the wannacry ransomware* [Internet web page]. Retrieved from <https://www.symantec.com/connect/blogs/what-you-need-know-about-wannacry-ransomware>
- TechTimes. (2014, 25September). *Apple denies icloud, find my iphone security breach: Only very targeted attacks* [Internet web page]. Retrieved from <http://www.techtimes.com/articles/14717/20140907/apple-denies-icloud-find-my-iphone-security-breach-only-very-targeted-attacks.htm>
- TheGuardian. (2015, March). *Uber denies security breach despite reports of logins for sale online* [Internet web page]. Retrieved from <http://www.theguardian.com/technology/2015/mar/30/uber-denies-security-breach-logins-for-sale-dark-web>
- TheGuardian. (2017). *'petya' ransomware attack: what is it and how can it be stopped?* [Internet web page]. Retrieved from <https://www.theguardian.com/technology/2017/jun/27/petya-ransomware-cyber-attack-who-what-why-how>
- TheRegister. (2017). *Strange mirai botnet brew blamed for powerful application layer attack* [Internet web page]. Retrieved from https://www.theregister.co.uk/2017/03/29/mirai_variant/
- TheRegister.co.uk. (2014). *Slap for snapchat web app in snap mishap flap: '200,000' snaps sapped* [Internet web page]. Retrieved from http://www.theregister.co.uk/2014/10/10/new_{}_photo_{}_hack_{}_claim_{}_200000_{}_snapchat_{}_photos/
- Thornburgh, T. (2004, October). Social engineering: The 'dark art'. In *ACM infoseccd conference'04* (p. 133-135). Kennesaw, GA, USA: ACM.
- Tong, J., Xiong, G., Zhao, Y. & Guo, L. (2013). A research on the vulnerability in popular p2p protocols. In *2013 8th international conference on communications and networking in china (chinacom)* (p. 405-409).
- TrendMicro. (2017). *Ransomware* [Internet web page]. Retrieved from <https://www.trendmicro.com/vinfo/us/security/definition/ransomware>

- Turpe, S. (2009, September). What is the shape of your security policy? security as a classification problem. In *ACM NSPW'09* (p. 23-36). Oxford, United Kingdom..
- Vahidov, R. (2006). Design researcher's is artifact: A representational framework. In *First international conference on design science research in information systems and technology (DESRIST 2006)* (p. 19-33). Claremont.
- Vaishnavi, V. & Kuechler, W. (2015). *Design science research in information systems* [Internet web page]. Retrieved from <http://www.desrist.org/design-research-in-information-systems/>
- Vaquero, L. M., Rodero-Merino, L. & Moron, D. (2011). Locking the sky: a survey on iaas cloud security. *Computing*, 91(1), 93-118. doi: 10.1007/s00607-010-0140-x
- Venable, J. (2006). A framework for design science reseach activities. In *2006 information resource management association conference*. Washington DC, USA.
- Vikas, S., Gurudatt, K., Pawan, K. & Shyam, G. (2014, February). Mobile cloud computing: Security threats. In *2014 international conference on electronics and communication systems (ICECS-2014)*. Coimbatore, India.
- Vom, B. J. & Buddendick, C. (2006). Reusable conceptual models - requirements based on the design science research paradigm. In *First international conference on design science research in information systems and technology (DESRIST 2006)* (p. 576-604). Claremont.
- Walls, J., Widmeyer, G. & Sawy, O. E. (2004). Assessing information system design theory in perspective: How useful was our 1992 initial rendition. *Journal of Information Technology Theory and Application*, 6(2), 43-58.
- Walls, J. G., Widmeyer, G. R. & Sawy, O. A. E. (1992). Building an information system design theory for vigilant eis. *Information Systems Research*, 3(1), 36-59.
- Webster, J. & Watson, R. T. (2002, June). Analyzing the past to repace for the future: Writing a literature review. *MIS Quarterly*, 26(2), xiii-xxiii.
- Wilson, P. (2011). Positive perspectives on cloud security. *Information Security Technical Report*, 16, 97-101.
- Wilson, W. (1999). Writing effective natural language requirements specifications. *Crosstalk J. Def. Softw. Eng.*, 16-19.
- Winter, R. (2008). Design science research in Europe. *European Journal of Information System*, 17, 470-475.
- Wired. (2017). *Hackers are trying to reignite wannacry with nonstop botnet attacks* [Internet web page]. Retrieved from <https://www.wired.com/2017/05/wannacry-ransomware-ddos-attack/>
- Wu, X., Zhanga, R., Zeng, B. & Zhoua, S. (2013). A trust evaluation model for cloud computing. *Procedia Computer Science*, 17, 1170-1177. doi: 10.1016/j.procs.2013.05.149
- Xie, B., Kumar, A. & Agrawal, D. P. (2008). Secure interconnection protocol for integrated internet and ad hoc networks. *Wireless Communications and Mobile Computing*, 8, 1129-1148. doi: 10.1002/wcm.557
- Xin, Z., Song-qing, L. & Nai-wen, L. (2012). Research on cloud computing data security model based on multi-dimension. In *2012 international symposium on*

- information technology in medicine and education* (p. 897-900). IEEE.
- Xu, X. (2012). From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28, 75–86.
- Yan, X., Zhang, X., Chen, T., Zhao, H. & Li, X. (2011). The research and design of cloud computing security framework. *Advances in Computer, Communication, Control & Automation, LNEE 121*, 757-763.
- Yang, J., Wang, C., Yu, L., Liu, C. & Peng, L. (2012). Network security evaluation model based on cloud computing. In *Icica 2012, part ii* (Vol. CCIS 308, p. 488-495). Springer-Verlag Berlin Heidelberg.
- Yao, A. C.-C. (2013, January). Classical physics and the church–turing thesis. *Journal of the ACM*, 50(1), 100-105.
- Younis, Y. A., Kifayat, K. & Merabti, M. (2014). An access control model for cloud computing. *Journal of Information Security and Applications*, 19, 45-60. doi: <http://dx.doi.org/10.1016/j.jisa.2014.04.003>
- Yu, H., Powell, N., Stenbridge, D. & Yuan, X. (2012, March). Cloud computing and security challenges. In *ACM SE*. Tuscaloosa, AL, USA: ACM. doi: 10.1145/2184512.2184581
- Zahran, J. (2014, May). *Holistic security: The risk wheel* [Internet web page]. Retrieved from <http://www.pinkerton.com/blog/holistic-security>
- Zhai, E., Chen, R., Wolinsky, D. I. & Ford, B. (2013, November). An untold story of redundant clouds: Making your service deployment truly reliable. In *Hotdep - workshop on hot topics in dependable systems*. Nemaquin Woodlands Resort, PA, USA: ACM.
- Zhang, G., Yang, Y. & Chen, J. (2012). A historical probability based noise generation strategy for privacy protection in cloud computing. *Journal of Computer and System Sciences*, 78, 1374–1381.
- Zissis, D. & Lekkas, D. (2012). Addressing cloud computing security issues. *Future Generation Computer Systems*, 28, 583–592.
- Zonouz, S., Houmansadr, A., Berthier, R., Borisov, N. & Sanders, W. (2013). Seccloud: A cloud-based comprehensive and lightweight security solution for smartphones. *Computers & Security*.

Appendix A

Glossary

Application Node When a default node accepts application components from other nodes, it becomes part of the family of the nodes that collectively deploy the application. Upon request, a default node may receive random components from other nodes. An application node may contain components for only one application.

BEC Back-end CIOS (see CIOS below).

CC Architecture CC architecture refers to the network architecture including in-premise hardware and their underlying platforms (i.e. software tools, OS). The architecture for a CC depicts how various components are arranged and interconnected. It may also depict the interface to the outside world (for example, connectivity to the public network infrastructure).

CIOS CIOS is the term given to the instances of OSs or platforms used within a Cloud architecture. CIOS may come in different flavours in a Cloud architecture: it may be installed on a computer/server as a stand-alone OS, or it may be installed as a VM on a type-1 or type-2 hypervisor.

Cloud Computing CC is a conceptual computing approach that may encapsulate any other computing means and act as a wrapper for all computing practices. CC is

the setting where hard (e.g. network infrastructure) and soft (e.g. data, software, processing) elements are remotely existent and access to these resources is on an ad-hoc basis using public or private communication infrastructure, where the management and maintenance concerns of the Cloud infrastructure including the resources held within the infrastructure are most often beyond the end-users' scope.

Cloud Security Model Cloud security model refers to the arrangement and provision within the Cloud architecture to ensure a safe computing environment. The security model may influence the architecture by implying specific provisioning and arrangement of network elements in a cloud architecture. In conjunction with the above, a security model incorporates specific security software and/or security mechanisms. A security model may be illustrated by means of a reference architecture for CC.

Cloud Service provider The business or entity that provides Cloud infrastructure and associated services to the Cloud users.

Community Cloud The Cloud that is shared by more than one organisations. This kind of Cloud is intended to be used by a community of organizations or entities having common concerns, interests or goals to be accomplished by the use of the Cloud.

Decentralised The components of an application are distributed in such a manner that there is no single 'core' of the system. The management and functionality of the systems are scattered and there is no single controlling entity (e.g. node) within the system. Each components of a decentralised system is equally powerful (or powerless) compared to any other node.

Default Node A Ki-Ngā-Kōpuku node that has all elements of a node except application component(s). A default node is ready to take any application components to turn itself into an application node. Databases in a default node are empty with

no information.

Distributed The components of an application are distributed among different servers.

A replication into different servers may mean distribution and thus it differs from decentralisation in such that decentralisation is essentially a distribution, but not necessarily vice versa.

Exception Flag Log The log file for exception or security alarms.

FEC Front-end CIOS.

Human Factors Human-centric actions that threaten a Cloud infrastructure.

Hybrid Cloud A mix of any of Public, Private or Community Clouds (at least two).

The Clouds forming a hybrid retain their own distinct characteristics, yet brings portability by means of load-balancing and options to switch among the Clouds of a hybrid Cloud.

IaaS The concept of IaaS entails the provision for a Cloud user to rent infrastructure from the CSP. In IaaS, the infrastructure is deemed to be a service to the end users and hence such name. Users are provided with Cloud resources to enable access to virtual server(s). As the infrastructure is provisioned in IaaS, the user can use the network, processing, storage, computers or servers with the facility of deploying any software set-up. However, in IaaS, only the infrastructure is provided; the platform or OS and the software that would reside on the platform remains under end-user ownership, implying the end-users are required to obtain and maintain them and any licensing issues.

Node A Ki-Ngā-Kōpuku node is the logical wrapper that contains the components of the application deployed. A CIOS may contain any number of nodes, and all the nodes do not necessarily contain identical components. Ki-Ngā-Kōpuku nodes have same structure but the application components held within the nodes are random.

PaaS The platform or the OS is provisioned to the client in PaaS model. In PaaS-associated Cloud services, the client does not have control over the platform but has total control of whatever can be achieved using that platform. A PaaS model allows a user to configure, deploy, install or un-install any software or service on the platform. In this approach, end-users do not need to worry about the infrastructure and platform licensing issues, but they need to obtain licenses and maintain software that would be installed on the platform. In PaaS, users can exploit the benefits of the platform but do not manage the platform and cannot access the Cloud infrastructure, network, storage and so on. PaaS has limited scope compared with IaaS.

Private Cloud The Cloud infrastructure that is operated and managed by private organisations or by an outsourced specialist third party. It can be either on premise or off-premise.

Public Cloud The Cloud infrastructure or services op for the public. It is normally operated and managed by a single CSP and services are open for subscription by Cloud users.

SaaS As the name implies, the user is able to use only the software provided by the Cloud provider and cannot extend control beyond the specific software the Cloud user is given access to. The access and management in SaaS are confined to the respective application or software for each Cloud user. Thus, in SaaS, the users are not authorised to manage resources at the IaaS or PaaS level. SaaS has limited scope compared to PaaS.

Virtualisation Virtualisation refers to the technology that enables resource sharing among different parties to reduce overall equipment and management costs

Appendix B

Ki-Ngā-Kōpuku Alternative Solution

B.1 Introduction

As part of future enhancement, some additional research is carried out. A modified version of Ki-Ngā-Kōpuku is presented here, where data security is also taken into account. It is important to note that the future enhancement discussed below is not complete research, and thus may not be taken as an optimised and integrated solution. There are areas that would require further research and improvement. The discussion below proposes an alternative version of Ki-Ngā-Kōpuku. The version of the proposed discussed in the thesis may work within the context of the version that is presented in this section.

B.1.1 Assumptions

In future development, the distribution may take place at CIOS level instead of at node level. At this level, provision is kept to mark a CIOS as either ‘Trusted’ or ‘Blacklisted’. Any suspicious CIOS will be blacklisted and will be taken out of Assumptions: context. The following proposed future development of the security model is based on the assumption that the security model may span among different Clouds. Thus the high

level communication will take place among the CIOs residing in different Clouds, not among the nodes. Further distribution of interaction among the components within a CIO would then be coordinated by the CIO. In this case, a CIO would merely work as bridge between its own nodes and the nodes from other clouds.

B.2 Security Mechanism

All the processing steps discussed here collectively forms the security mechanism for Ki-Ngā-Kōpuku. The processing is illustrated with the assumption of one application being deployed. The process of deployment universal, that is, same for any application.

B.2.1 Componentisation

The componentisation of an application is application specific. Ki-Ngā-Kōpuku defines the generic steps for componentisation. For any application, it is important to decide total functionality of the respective application as well as the data requirements. Apart from these, total number of requests that could be initiated by the end-users needs to be listed. Thus, the following three components should be decided before componentisation of an application:

- Functionality
- Data
- End-user request

B.2.2 The first CIO

There is a starting point of the architecture and initially there is one CIO within BEC premise. This is assumed as the ‘originating BEC’. Once the originating BEC is in place, adding at least one new CIO will complete the BEC scenario where component

distribution and other functionalities are automated process – this is discussed in section B.3 on page 217. However, the originating BEC will consist of all the components and modules to carry out other functionalities. That is, in addition to the components themselves, the originating CIOS will contain mechanisms for adding a new CIOS and component distribution as well as mechanisms for inter-component interfacing and incident monitoring. These process are discussed later in this chapter. Apart from the above, the originating BEC would contain a number of databases as below:

Component count: This is the list of the total number of components in the BEC premise. The BECs will keep track of the total number of existing components distributed so far. Such tracking is to ensure that the component distribution is done in a nearly balanced manner and no single component outweighs in number compared to other components. For example, if an application has three components, say λ_1 , λ_2 and λ_3 ; and if the number of these components at any given time are respectively 35, 34 and 37, then the distribution of components can be considered as a nearly balanced one. However, if the existing components number were, let's say 30, 24 and 76, then it would be an imbalanced distribution. Keeping a track of total number of existing component would help a nearly balanced distribution. The mechanism of 'component count' is discussed in Section B.3 on page 221 under 'component Selection' process. This file also keep two additional parameters: total number of components and total number of components to be distributed where the later one is always less than the former one.

Trusted CIOS: This is the list of the BECs that are trusted within BEC premise. All the BECs acknowledge and respect all requests only from the trusted CIOSs.

CIOS blacklist: This is the blacklist of the CIOSs that must be ignored. Once a CIOS is blacklisted, it is ignored by all the trusted CIOSs. A CIOS gets blacklisted

upon any suspicion of malicious or inconsistent activities. This is illustrated in later discussion where various processing approaches are explained.

‘The chosen one’ list: This is the list of the BECs that are chosen to deal with FECs. Upon processing completion, one of the BECs would require to send processing. For this, the BECs that are involved in processing that particular processing choose one BEC to send output back to FEC – this BEC is marked as ‘the chosen one’. It is required to keep a list of the chosen one so that one specific BEC is not being chosen successively to ensure random scenario and to eliminate the probability of one BEC taking control of this specific task. This mechanism is illustrated in later discussion where various processing approaches are explained.

Exception flag log: This is the log file that records all exceptions. This is illustrated in later discussion where various processing approaches are explained.

Distribution count: This is used only by the originating BEC. The originating BEC destructs itself after a certain number of components are distributed. To accomplish this, the originating BEC keeps track of the number of distribution to determine the point to self-destruct. This is explained below and the mechanism is illustrated in Section B.2.2 on page 216, and in Figure B.2 on page 216.

In addition to the above, the originating BEC also contains a self-destructing mechanism. Figure B.1 on the following page illustrates the elements that the originating BEC holds. The elements marked with asterisk (*) are held only and only in the originating BEC and not inherited in any other BECs.

All the CIOs that are added gradually inherits all the elements depicted in Figure B.1 on the next page, except the exact number of components, the self-destructing mechanism and the ‘Distribution Count’ database. A newly added CIO may or may not have exactly same number of components. This is because the components are

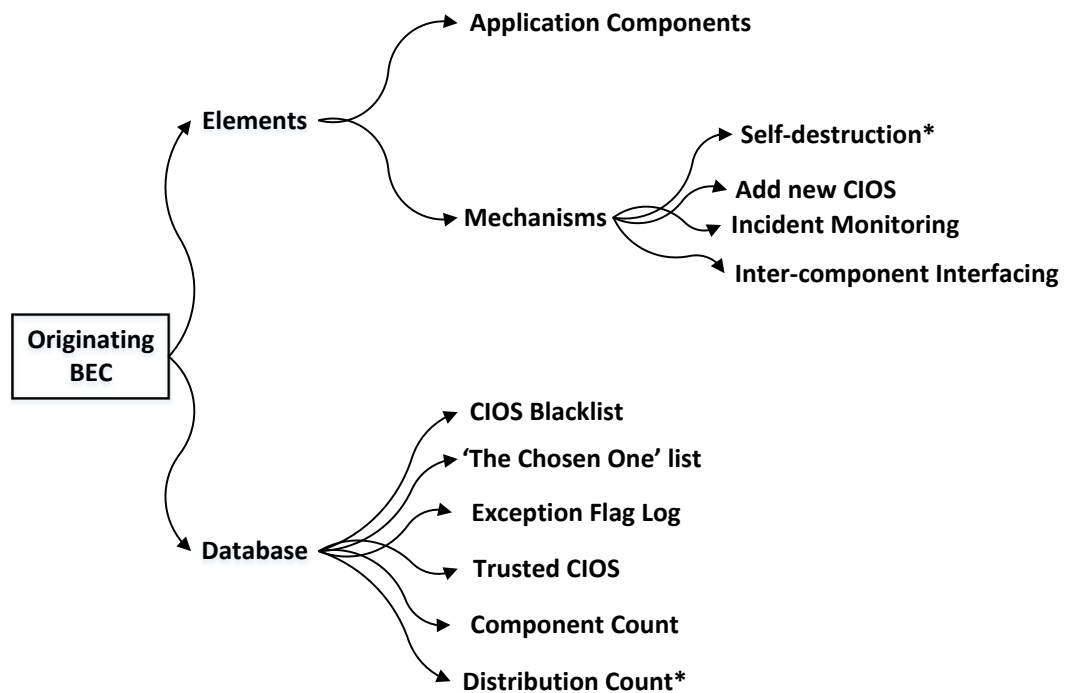


Figure B.1: Elements of the Originating BEC

randomly distributed among all the BEC so a BEC may not essentially hold the exact components of any other given BEC. The self-destructing mechanism is also solely deployed only and only in the originating BEC so that the originating BEC become unavailable after a certain number of distribution to ensure there is no single BEC that holds all the components of an application. This is due to the fact that initially a CIOS needs to contain all the components of an application to initiate the BEC scenario of Ki-Ngā-Kōpuku. To keep track of the number of distribution, the originating BEC uses 'Distribution count' database as mentioned above. Thus, the 'Distribution Count' database does not exist in any BEC except the originating BEC. The self-destruction mechanism for the originating BEC is discussed below.

Self-destruction Mechanism

As discussed earlier, any given BEC within Ki-Ngā-Kōpuku may not hold all the components of an application to reflect random scenario. But it is impossible to distribute all the components if the originating BEC does not hold all the components. At the same time, to ensure the above criteria, the originating BEC would destroy itself when all the components are distributed in other added BECs. Upon distributing all the components to other added BECs, the originating BEC can destruct itself yet the components will be residing in other BEC but no single BEC will hold all the components – this is ensured through ‘Component Selection’ process and discussed in Section B.3 on the following page. Figure B.2 illustrates the self-destruction steps.

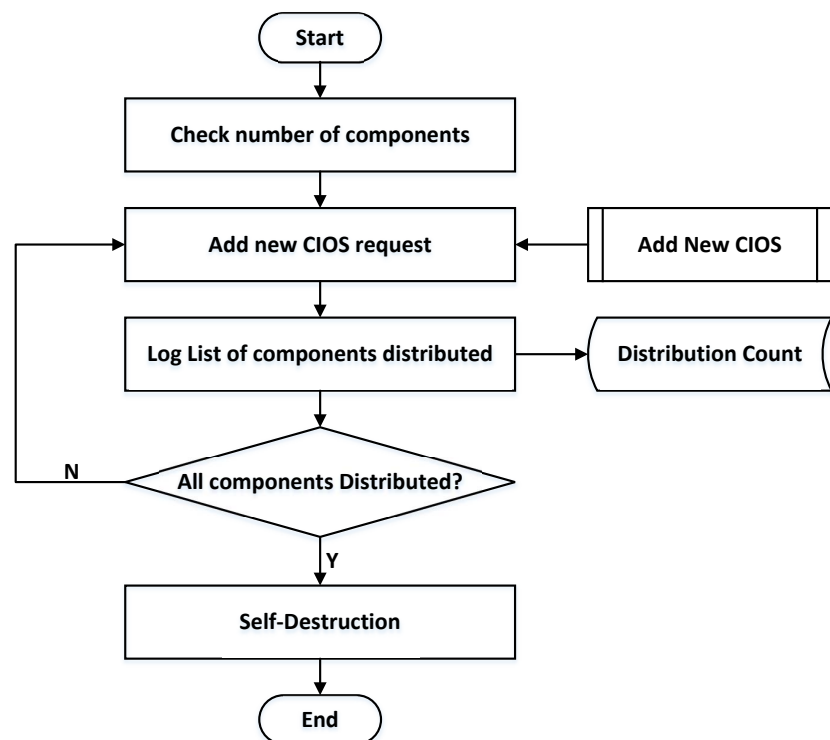


Figure B.2: Self-destruction Mechanism for Originating BEC

As illustrated in Figure B.2, the originating BEC first keeps track of total number of components. When a new CIOS is added as BEC, some components are distributed to

the new BEC (through ‘Component Selection’ process discussed later in Section B.3). At this point, some components are distributed but not all. The same repeats with other new BECs but every time with different components. The originating BEC keep track of the components distributed and log them in ‘Distribution Count’ file. After few iterations, all the components gets distributed in other BEC where the originating BEC is the only one that holds all the components in it. At this point, the originating BEC can be taken down yet all the components will be existent in the BEC premise in a random manner. Thus, upon distributing all the components to the BECs, the originating BEC destroys itself and the randomness of component distribution from this point gets consistency.

B.3 Add New CIOS

At BEC level of the architecture, the scenario is random where a new CIOS may come ‘alive’ at any time. When a new CIOS is comes alive, the existing BECs need to make it part of the BEC family. This is done by distributing components from the existing BECs to the newly added server. The flowchart of the steps to add a new CIOS are illustrated in Figure B.3 on the following page.

Once a new CIOS become ‘live’ within BEC premise, it will broadcast a request to the existing CIOS to make it a trusted BEC and to eventually receive application components to work as a BEC. It is assumed that the new CIOS is the ‘requestor’. Upon receiving the request, existing BECs randomly broadcast components to the requestor – this is where the component distribution takes place before going to the next step of the flowchart in Figure B.2 on the previous page. The component distribution is discussed later and illustrated in Figure B.3 on the following page.

Once the component distribution is done, the requestor creates ‘key’ from the received components and broadcasts the key. The creation and sharing of key helps

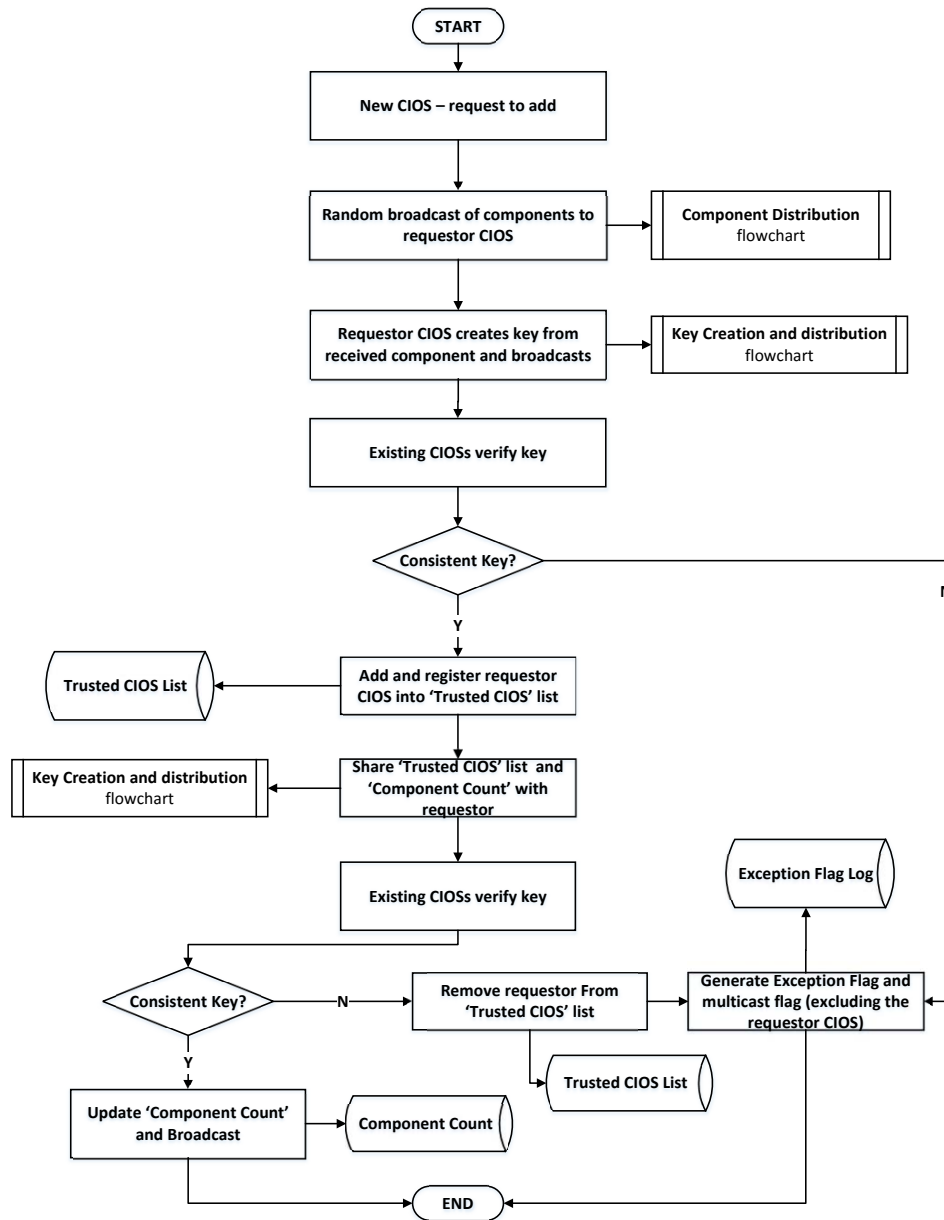


Figure B.3: Add New CIOs

to determine the BECs that the requestor has received the authentic components. Key creation and distribution is discussed later in Section B.3 on page 223 and illustrated in Figure B.4 on page 220.

Once the requestor creates key and distributes the key to the BECs, the key is

checked for consistency. If the key is inconsistent, an exception flag is generated by the BECs and shared among the BECs without informing the requestor. Subsequently, the requestor is excluded from making it a BEC. The exception flag triggers another process that is discussed later and not part of this 'Add New CIOS' process. This initiates the incident monitoring process to take care of the requestor with inconsistent components.

However, if the distributed key found to be consistent in 'Key Creation and Distribution' process, the requestor becomes a BEC and is entered into the 'trusted CIOS' list. At this point, it is assumed that the CIOS has become the 'new BEC'. The existing BECs then share the trusted CIOS list with the new BEC and the new BEC randomly picks one list and keep it. The key creation and distribution takes place at this point for one more time to ensure the new BEC holding the authentic list of trusted CIOSs. If the key distribution of the trusted CIOS list is consistent, the process ends successfully. If not, an exception flag is generated as explained before and as illustrated in Figure B.2 on page 216; the generation of exception flag at this stage would also associate the new BEC being removed from the trusted CIOS list database.

Once a new CIOS becomes a BEC and components are successfully distributed in the new BEC, the 'Component Count' database is updated where the number of components distributed gets incremented in 'Component Count'. To illustrate this with an example, let the components be λ_1 , λ_2 and λ_3 and the component count is $\lambda_1=12$, $\lambda_2=10$ and $\lambda_3=10$. Upon adding a new CIOS, the 'Component Selection' process decides to distribute λ_2 and λ_3 . Once the distribution is done the requestor CIOS becomes a BEC, the number of the distributed components would be increased by 1, that is, $\lambda_2=11$ and $\lambda_3=11$ would be the updated component count. This needs to be updated in the 'Component Count' list once a requestor CIOS is added successfully and it becomes a new BEC. Once update, the component count now should be $\lambda_1=12$, $\lambda_2=11$ and $\lambda_3=11$.

Component Distribution

Figure B.4 illustrates component distribution which is a sub-process of adding a new CIOS.

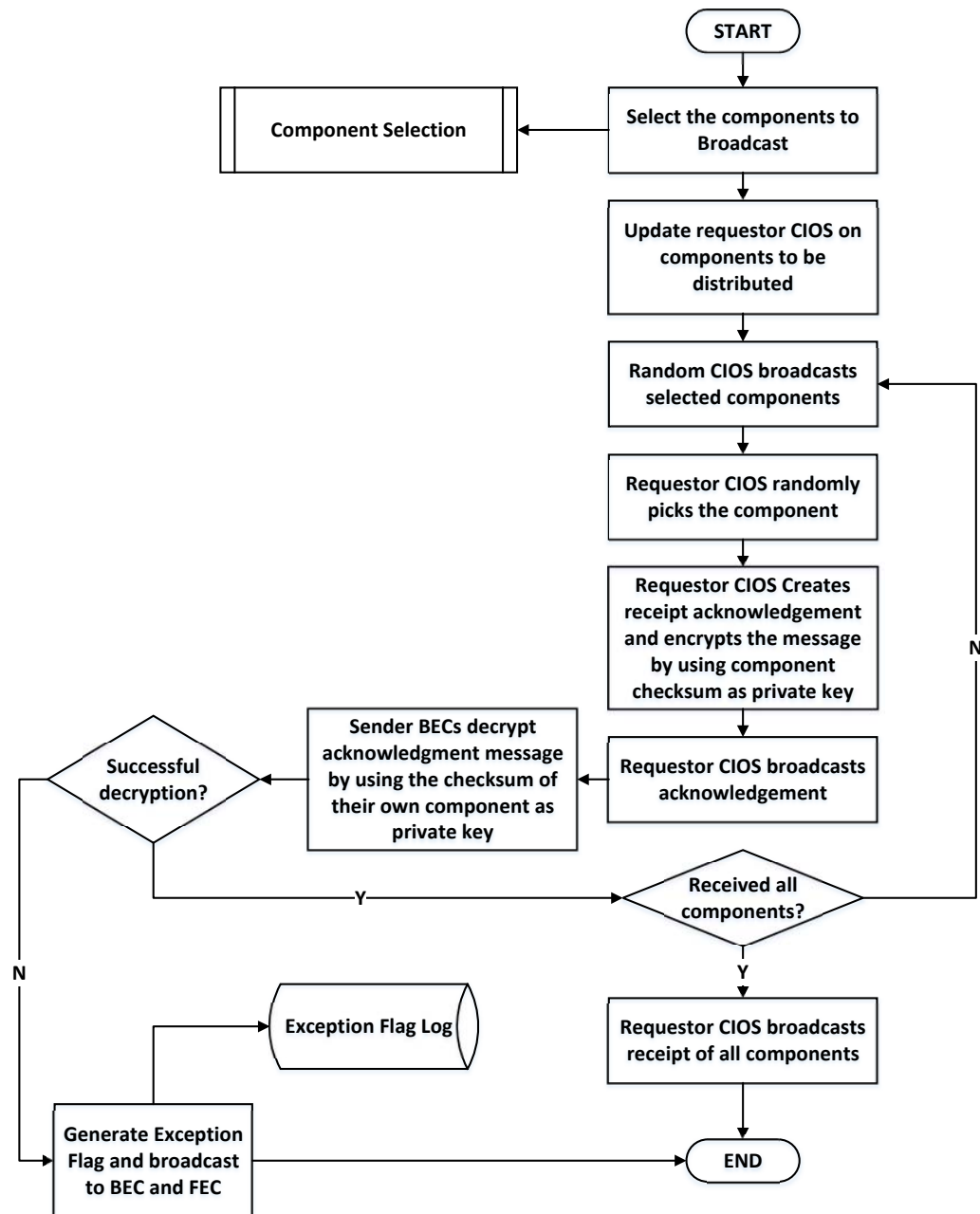


Figure B.4: Component Distribution

Before components can be distributed, the components to be distributed must be decided upon. To ensure that a single BEC does not contain all components, those distributed are selected. This in turn is to ensure that any given BEC do not hold all the components of an application. A sub-process named 'Component Selection' determines the components to be distributed. The requestor is then updated on the components that is it to be received. Random BECs then broadcast the selected components. The requestor randomly picks the components from the broadcast. For all received components, the requestor creates a checksum of the component received and uses this checksum as the private key to encrypt a standard acknowledgement message. The requestor then encrypts the acknowledgement message with the private key and broadcasts. Existing BECs receive this broadcast encrypted acknowledgement message. Since the acknowledgement is encrypted with the checksum of the received components by the requestor and since the existing BECs hold the same components, the checksum created locally by the BECs should match the checksum created by the requestor. This, the requestor does not need to convey any knowledge on the private key yet the BECs must be able to decrypt the acknowledgement message by using their own created checksum. Any manipulation of the components at the requestor end would result in mismatching checksum and thus the BECs would not be able to decrypt the acknowledgement message using their own checksum. In this case, an exception flag will be generated by the BECs and broadcast. Otherwise, the component distribution would end successfully.

Component Selection

Figure B.5 on the next page illustrates component selection which is a sub-process of component distribution.

Total number of components for an application is fixed by the developers during the componentisation phase, so is the total number of components to be distributed

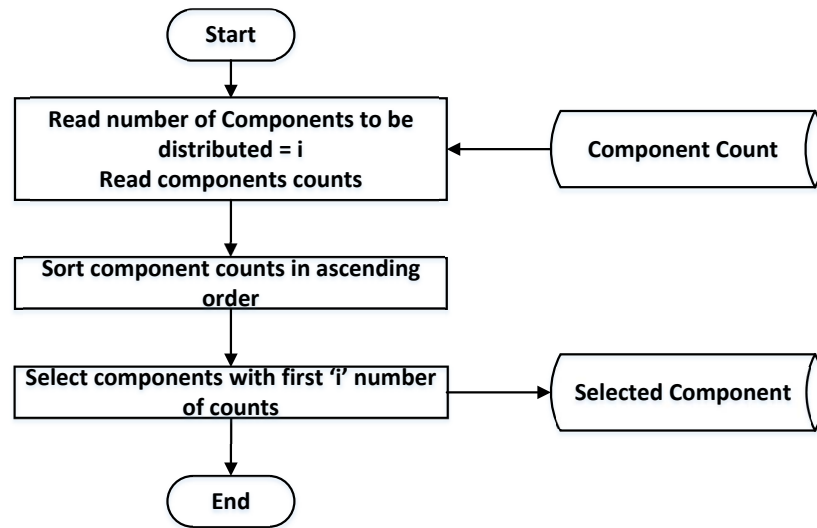


Figure B.5: Component Selection

in one distribution while adding a new CIOS. For example, if an application has five components, the developers may decide to distribute only three components at any given distribution. This will prevent one BEC to have all the components. At the same time, it needs to be ensured that the same components are not distributed successively. This is done and ensured by the Component Selection process as illustrated in Figure B.5. If i number of components are to be selected, the component selection process will read the component count file and then sort the component count in ascending order. Then the process will select the components associated with the first i number of counts from the sorted list.

The above is illustrated below with an example. It is assumed that there are three components named λ_1 , λ_2 and λ_3 ; and the component count is $\lambda_1=12$, $\lambda_2=10$ and $\lambda_3=10$. Let us also assume that the developers have decided to distribute two components for every distribution. Thus, here $i = 2$. The 'Component Selection' process will sort the component count and the list will $\lambda_2=10$, $\lambda_3=10$ and $\lambda_1=12$. Since $i=2$, the first two components from the sorted list (that is, λ_2 and λ_3) will be selected to be distributed to the newly added CIOS. This will ensure all the components are being distributed evenly

and no single component outweighs others in number at any given scenario within the BEC premise.

Key Creation and Distribution

Figure B.6 on the next page illustrates the process of key creation and distribution which is a way to ensure that all the BECs are holding the consistent version any data or output of any processing.

The ‘Key Creation and Distribution’ is a process to generate key from the output of another processing, to ensure every BEC involved in a processing are holding the same and consistent version of the output. This in turn will ensure to malicious parts of data or code are injected during processing within an application. As described in Chapter 5 on page 112, one component will be held in a number of BECs and upon processing request from the end-users, all the BECs that hold the component(s) associated with the processing request will process the request. Upon completion of the processing, one of the BECs will be randomly chosen to send the outcome back to the end-users end. Now, as there will be more than one BEC involved in processing the same, it is important to ascertain that no BEC is able to inject undesired data or code. For this, the involved BECs will produce checksum of the data or processing at hand and will use the checksum as private key to encrypt a standard message indicating the processing is complete. The checksum for all the involved BECs must be the same, since they worked on the same processing request. Thus, the encrypted message must be successfully decrypted by all involved BECs by their own produced checksum. The checksum used as private key must be same for all concerned BEC and thus the sharing of private key is not required.

Any inconsistency in any of the BEC would result in a different checksum compared to the rest of the involved BECs and thus the message with inconsistent checksum would not be possible to decrypt by other BEC. At this point an exception flag will be

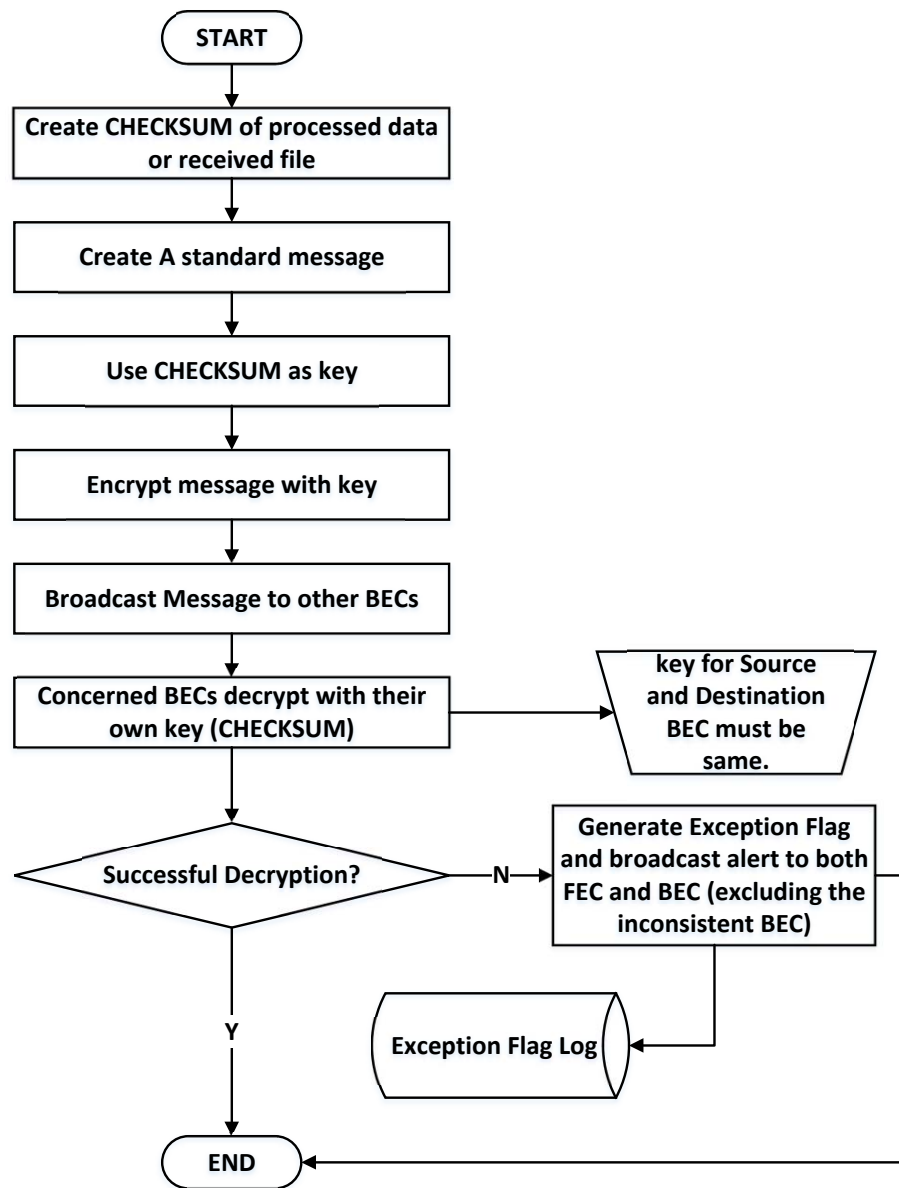


Figure B.6: Key Creation and Distribution

generated and broadcast.

B.4 Inter-Component Interfacing

Inter-component interfacing is required for the components residing in different BECs. Thus, this is essentially data transfer among the BECs (whether it is a processing request,

data or any other information sharing). The process of data/information sharing among the BECs are illustrated in Figure B.7.

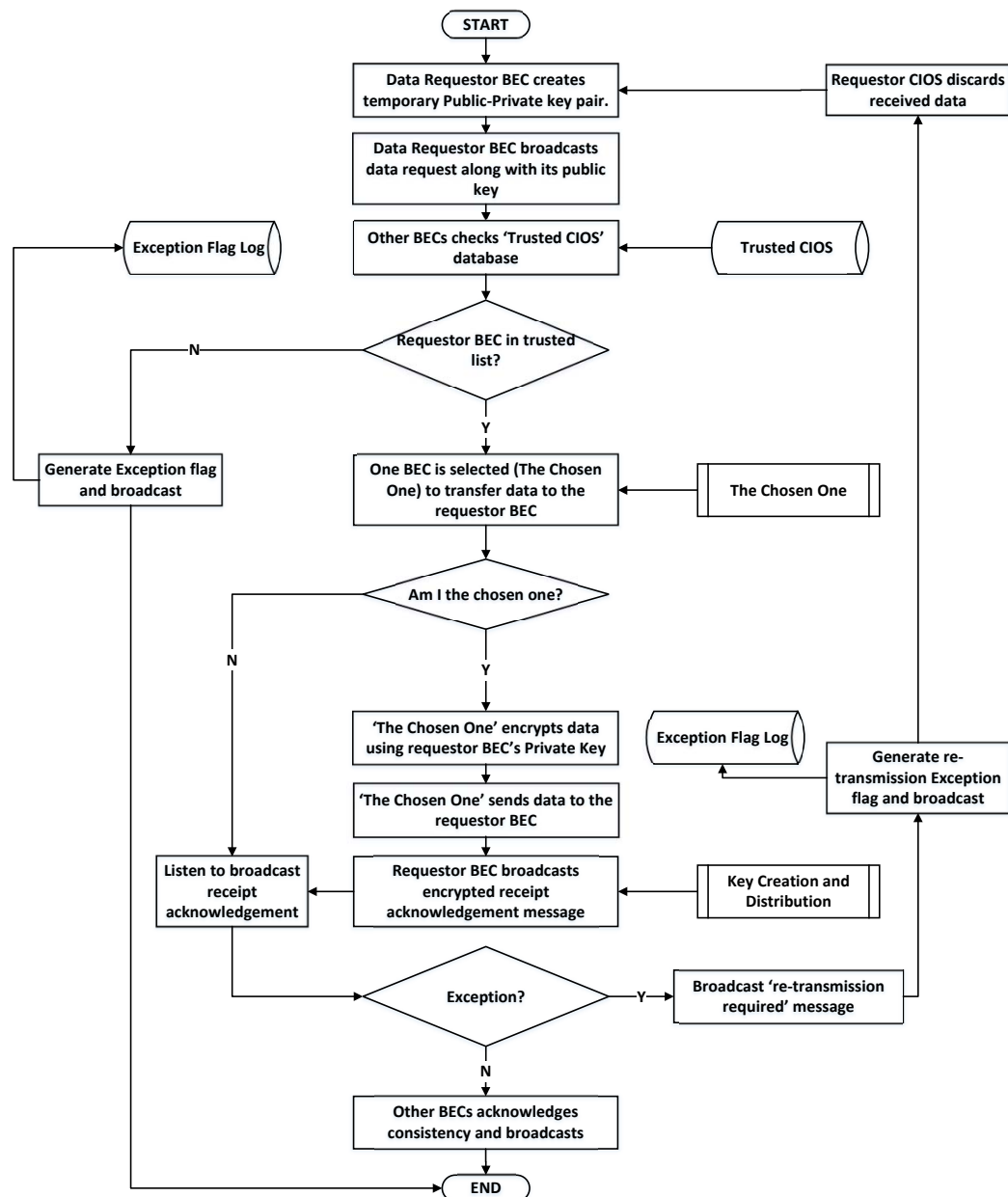


Figure B.7: Data Transfer among BECs

As illustrated, the process starts with a BEC requesting information/data from other BECs. To do that, the requestor BEC creates a temporary public-private key pair (for

asymmetric encryption). This key pair is valid only for a single request and there must be a new key-pair generated for every new request made. The request to get data/information also includes the public key created in the above step. Upon receiving the request, other BECs check whether the requestor BEC is in the 'Trusted CIOs' list. If not, an exception flag is generated and the ends with exception.

If the requestor BEC is in the trusted list, other BECs respect the request to proceed with. One of the BECs is randomly chosen to send the data/information to the requestor BEC. To randomly choose one BEC to send data to requestor, a sub-process named 'The Chosen One' is executed. 'The Chosen One' process is discussed below. As it will be seen, 'The Chosen One' process ensures that a single BEC is not being chosen successively for consecutive requests, to maintain desired randomness. However, once a random BEC is chosen to send data/information to the requestor, it becomes the chosen one. The chosen one then encrypts the data/information using the requestor BEC's sent public key, and sends the data/information to the requestor BEC. Upon receiving the data/information, the requestor BEC creates and encrypted acknowledgement message using the earlier explained sub process 'Key Creation and Distribution' and broadcasts the message. If the execution of the sub-process "Key Creation and Distribution" yields no exception, it would mean the send data/information is received in consistent state and thus the process ends. On the other hand, the generation of an exception at this stage would indicate reception of inconsistent data/information by the requestor BEC, and an exception flag is generated. This will trigger a message to re-start the process broadcast by the BECs and the requestor BEC would discard received data and start the request process again with a new key pair.

The Chosen One

'The Chosen One' is the process to select one BEC in the scenarios where more than one BEC is involved in doing the same task. By the way Ki-Ngā-Kōpuku works, it is a

common scenario for a number of BECs working on the same task, but the recipient of such task could to be sent the output/outcome only by one BEC. In this kind of situations, it needs to be assured that no single BEC may becomes the permanent messenger. To ensure this, every time a new BEC will be chosen to convey the output/outcome of any task back to other BECs or back to FEC. ‘The Chosen One’ process will ensure that every time a new BEC is randomly chosen and no single BEC is chosen successively. To keep track of the chosen BEC to act as the chosen one, Ki-Ngā-Kōpuku maintains a file in all the nodes. This file can be called ‘The Chosen One’ (the same name as the process itself) which will hold the list of the BECs that have been chosen in previous tasks. The steps of ‘The Chosen One’ process is illustrated in Figure B.8 on the following page.

The first step for a BEC is to read the chosen one file to ensure if the processing BEC was the last chosen one. If so, it does not send a request to be a contestant to become the chosen one. If not, it will broadcast a contestant request as a nominee to be the chosen one. At the same time, it listens to the broadcast contestant request from other BECs. If any of the contestant request comes from the last chosen one BEC, it generates an exception flag and marks the respective BEC as an inconsistent one. It then randomly chooses one BEC from the ‘Trusted CIOS’ list but before doing so, it excludes the previously identified inconsistent BEC from the trusted list. It then broadcasts its selected one as its vote. The voting process starts at this point which is a sub-process for ‘The Chosen One’ process, which returns a selected BEC as the chosen one as the outcome of the voting process. The ‘Voting’ process is discussed below.

Voting

The voting process takes place to select one BEC to deal with transmission/transfer of data/information. This process is illustrated in Figure B.9 on page 229.

The process starts with listening to all the broadcast ‘vote’ and keeping them in memory. The first vote is then picked up and checked whether the vote came from a

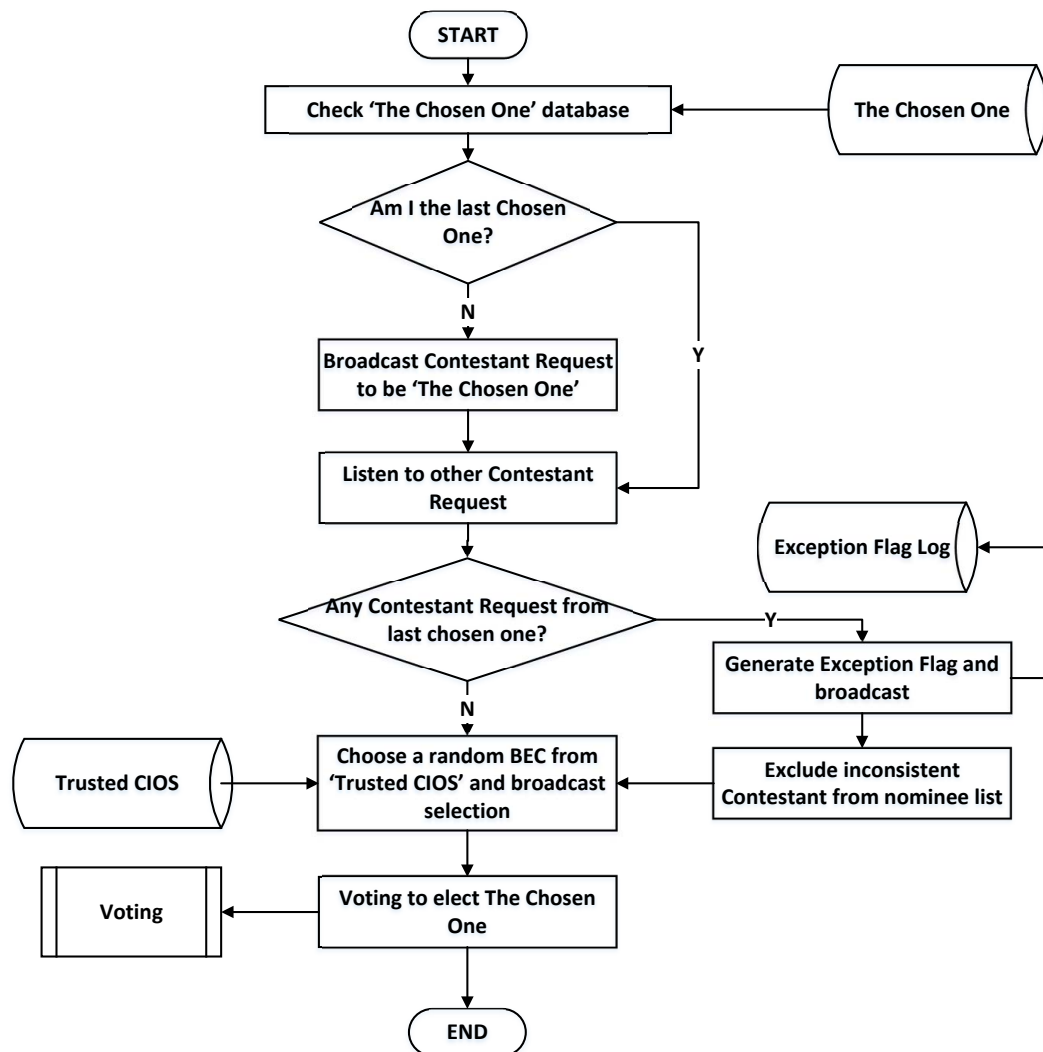


Figure B.8: The Chosen One

trusted BEC. The sender is compared against the 'Trusted CIOS' list to ascertain this. A vote cannot come from any BEC that is not in the trusted list. So, a vote originating from a source that is not in the trusted list would result in generation of an exception flag and being broadcast by the processing BEC, the vote is discarded and the next vote is read.

If the vote comes from a trusted BEC, the vote is counted and kept in a temporary file 'Vote Count'. Once all the votes are read and counted, the BEC with highest number of votes becomes the winner and thus becomes the chosen one, provided the winner

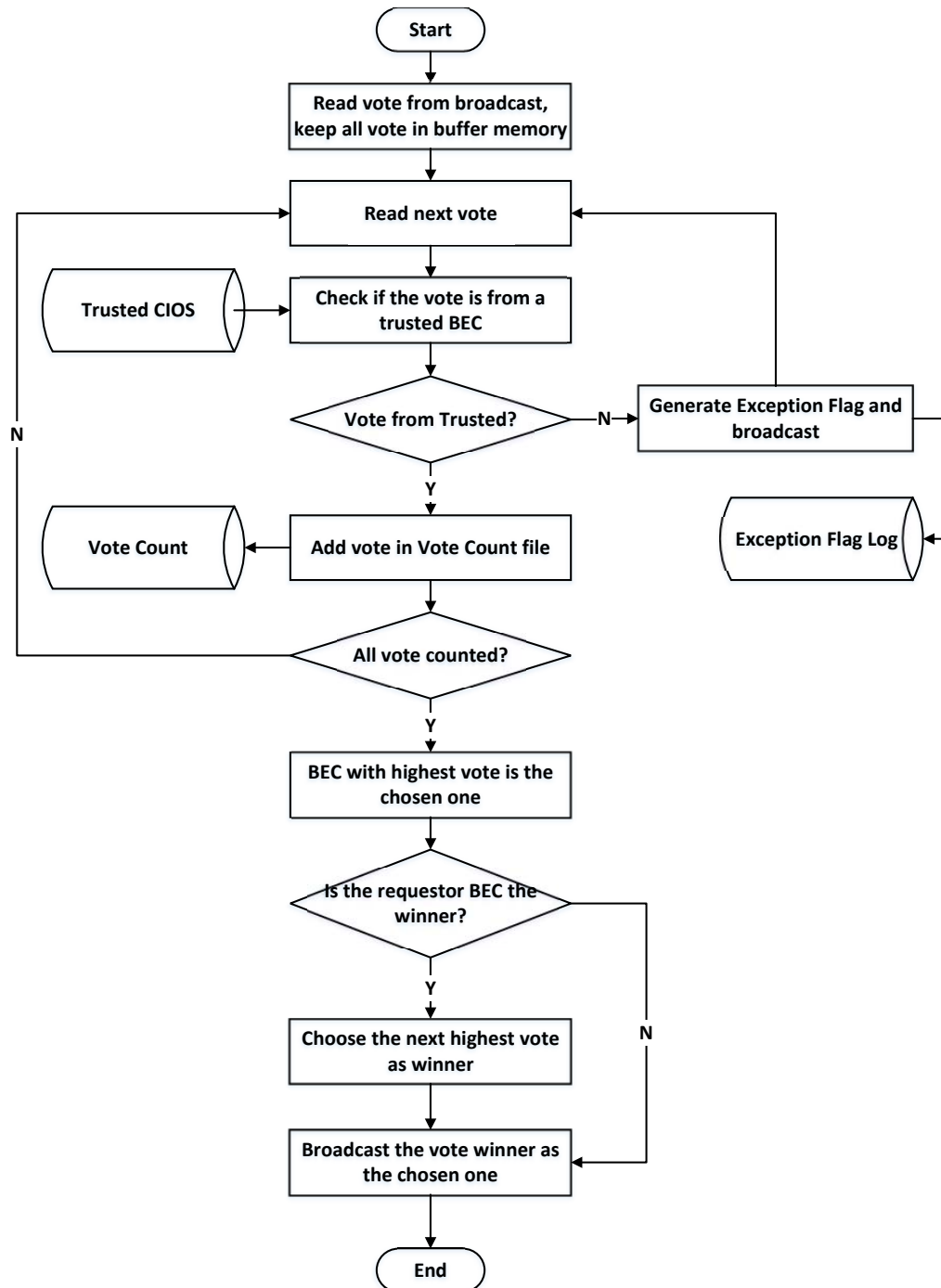


Figure B.9: The Voting Process

is not the 'requestor BEC' in the 'Data Transfer Among BECs' process. If the winner is the 'requestor BEC', then the BEC with next highest vote becomes the chosen one,

as illustrated in Figure B.9 on the preceding page. The name of the chosen one then is broadcast to other BECs.

B.5 Incident Monitoring

Incident monitoring involves monitoring for exception flags and taking action once an exception flag is detected. The processes described earlier suggest that, any inconsistency of suspicious act within the infrastructure would generate Exception Flag. The process of incident monitoring and response would come into action any-time an Exception Flag is detected. Upon detection of an Exception Flag, the BECs collectively take action to exclude the inconsistent BEC from any future processing or information sharing and blacklists the inconsistent BEC. There is only one exception where the BEC is not excluded – when the exception flag is a ‘Re-transmission Exception Flag’. All the BECs actively monitor for any exception flag generated and all the BECs collectively take action to exclude any inconsistent BEC from its trusted list. The process is illustrated in Figure B.10 on the next page.

All BECs actively monitor the environment and listens to the broadcast to detect the exception flag. When an exception flag is detected, it must be from more than one BEC. If the exception flag comes from only one BEC, then the BEC sending the exception flag is inconsistent and the exception flag is false alarm. The inconsistent BEC is then blacklisted and moved from trusted list to the blacklist.

If the generated exception flag is coming from more than one and different BECs, it is then checked whether it is a ‘Re-Transmission Exception Flag’. If this is the case, the exception flag is ignored. Otherwise, the BEC that caused the exception flag to be generated is blacklisted and moved from trusted list to blacklist.

Excluding an inconsistent BEC does not affect the operation of an application within Ki-Ngā-Kōpuku architecture, as there always exist a number of BECs with redundant

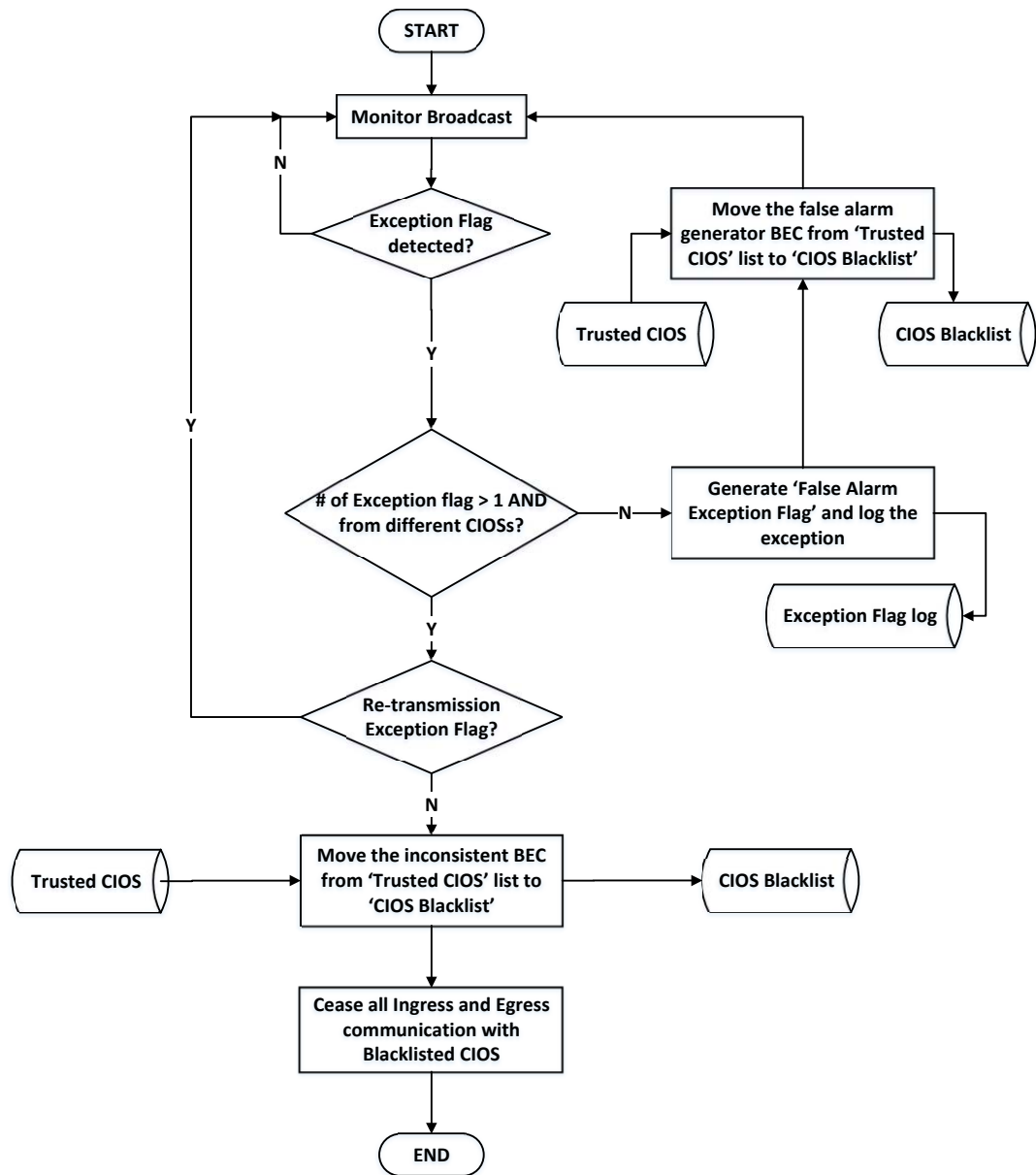


Figure B.10: Incident Monitoring

copies of everything that an inconsistent BEC may held. This is the mechanism by which Ki-Ngā-Kōpuku applies its self-healing feature.

Appendix C

CORP Algorithm

C.1 Introduction

One of the approaches used in the model explained in Appendix B on page 211 involves a number of nodes that collaboratively carry out required processing. The algorithm behind this is called COLlaborative Redundant Processing (CORP). CORP is discussed in further details in this section. CORP is an approach to prevent unauthorised modifications of information in Ki-Ngā-Kōpuku. CORP introduces redundancy where a more than one node carry out the same processing. Upon processing, the nodes collectively determine the consistency of the processed data. If a node gets compromised or acts in a malicious way, other nodes are able to detect. CORP is developed as a future add-on for Ki-Ngā-Kōpuku. Ki-Ngā-Kōpuku is an ongoing research which currently addresses application availability and prevents attacks, for example DDoS. Data security is considered as future development for Ki-Ngā-Kōpuku. CORP algorithm is envisioned to be a part of Ki-Ngā-Kōpuku in its future enhancement. In CORP, a number of servers/nodes carry out the same processing and it ascertains that no unwanted modification on the data is carried out. Any act with malicious intention to make unauthorised modification is detected as part of the steps of the algorithm. CORP is developed mainly

for a CC security model that has specific architectural requirement, but the concept can be used in any distributed and redundant computing.

C.2 Problem Context

At the back end of Ki-Ngā-Kōpuku, there will be a number of servers/nodes. Any application is chopped up into several components and scattered among the nodes in a random manner. The components are distributed among the nodes in such a manner that no node contains all the components, and there are multiple copies of any given components in different nodes. As a result, the components distribution in Ki-Ngā-Kōpuku associates redundant copies of components with no single point of failure.

Figure C.1 illustrates the high level view of a computing architecture where an application is chopped down into several components and distributed randomly among a number of nodes.

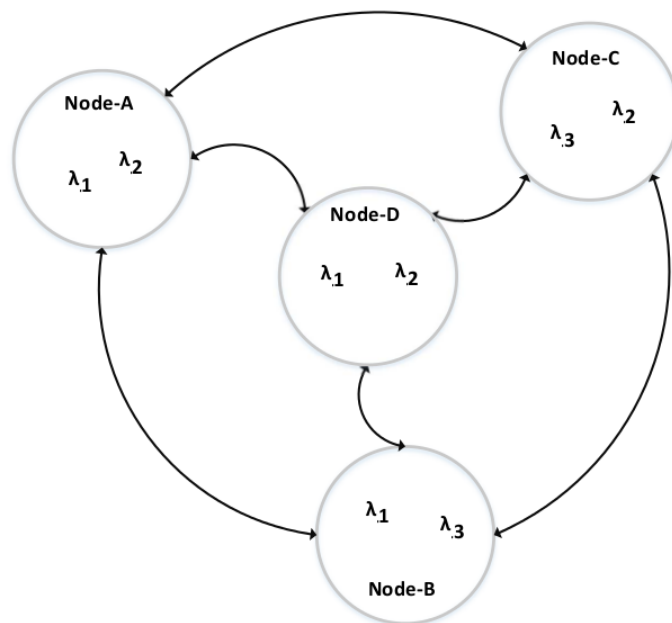


Figure C.1: Computing Architecture for CORP

Referring to Figure C.1 on the preceding page, it is assumed that an application is broken down into three components λ_1 , λ_2 and λ_3 . The components are randomly distributed to a random number of nodes. Three nodes are shown in Figure C.1 on the previous page for illustration purpose only. If the number of components are N , then the number of nodes must be greater than N . For example, if an application is made up of three components, there must be at least four nodes existence in the architecture. While it is the lower limit of the nodes, and there is no theoretical upper limit. How new nodes are added to the scenario, or how the components are randomly distributed to a new node are out of the scope of CORP.

The components are distributed in such a manner that no single node may contain all the components, and there must be redundant copies of each component. All the components collaboratively work within this architecture. There is no single point of failure as taking down one node will not make all the components unavailable due to the fact that the redundant copies are existent in other nodes. When a process is associated with a component for example λ_1 , all the nodes carry out the processing. For example, considering the scenario in Figure C.1 on the preceding page, if a process is associated with the component λ_1 , the processing takes place in Node-A, Node-B and Node-D but not in Node-C since the last one does not contain λ_1 . The nodes collaboratively ensure that no malicious data modification is carried out, which is achieved by the CORP algorithm discussed in Section C.3 on the next page. The nodes also ascertain collaboratively that all the nodes containing λ_1 end up with the same output ensuring data consistency is maintained and unauthorised data modification is prevented.

In the above scenario, it is imperative to justify the credibility of having redundant copies of a component; as well as how processing associated with a component can be carried out consistently. Also, the malicious acts of carrying out unauthorised operations/modification on the data within the components are also of concern. Such justification is done by applying CORP into the above computing architecture. Having

discussed the distributed and decentralised architectural scenario of Ki-Ngā-Kōpuku where CORP is to be deployed, Section C.3 explains the steps involved in CORP.

C.3 CORP Algorithm

While CORP's main purpose is to prevent unauthorised data modification, it takes other existing computing approaches as aide to carry out its operation. CORP may use approaches found in, for example, hashing, digital signature or checksum.

Now, when processing occurs within any component, all the nodes holding the same component will do the same processing. The concept of CORP is that, since more than one node will carry out the same processing, the result/outcome of the processing must be exactly same for all nodes. CORP defines the mechanism by which the nodes will be able to check integrity of the data in all the nodes, and at the same time, will be able to raise alarm if any node is compromised or trying any malicious act to modify the data being processed.

When CORP is used, all the nodes work collaboratively to determine whether data in any of the participating node is modified. CORP is initiated once the processing is carried out by all the concerned nodes. CORP takes places only in the nodes involved in processing, not all the nodes present in the architecture. The steps involved in CORP algorithm are illustrated in Figure C.2 on the next page.

When processing is done, all the nodes that took part in processing initiate the process of CORP algorithm to verify consistency and integrity of the data processed. Figure C.2 on the following page illustrates the steps take place in every node involved in processing. The node first creates checksum of the processed data using any standard checksum generator algorithm. The node then creates a standard pre-defined message and uses the created checksum as the key to encrypt the message. The encrypted message is then multicast to all other nodes that are involved in the same processing. At

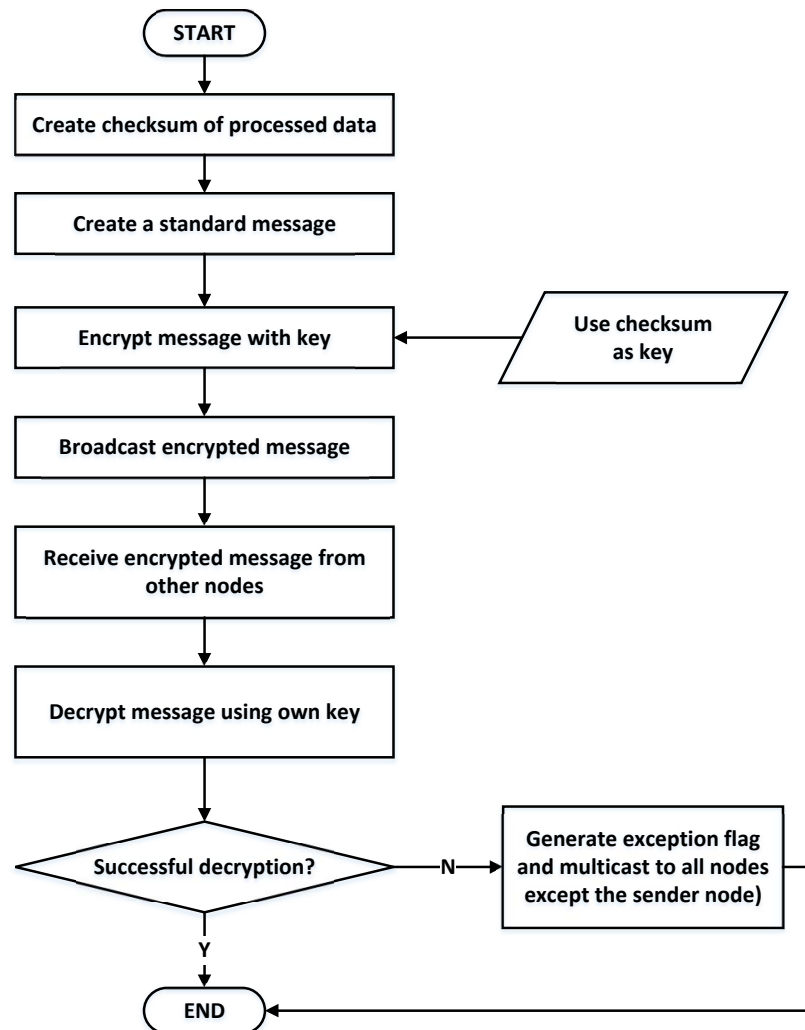


Figure C.2: CORP Algorithm

the same time, the node receives the same encrypted message sent from other concerned nodes and decrypts the message using its own checksum as key. The nodes do not need to share the key as all the nodes involved would generate exactly the same checksum if data integrity is not breached. Thus, successful decryption would mean no attempt of unauthorised data modification. If any message cannot be decrypted, it would indicate a possible attempt to modify data and the node would generate an exception flag as alarm. The exception flag is then multicast to all the nodes within the architecture including the ones that were not involved in processing.

If no exception flag is generated during the process, it can be decided that no attempt of unauthorised data modification took place. Otherwise, in the case of exception flag, other mechanisms of the security model come into action to decide what to do with the suspected node. However, this is not addressed in this paper, as the mechanism that will deal with a malicious node is out of the scope of CORP algorithm.

Figure C.3 illustrates the ideal scenario (i.e. assuming no exception) of CORP.

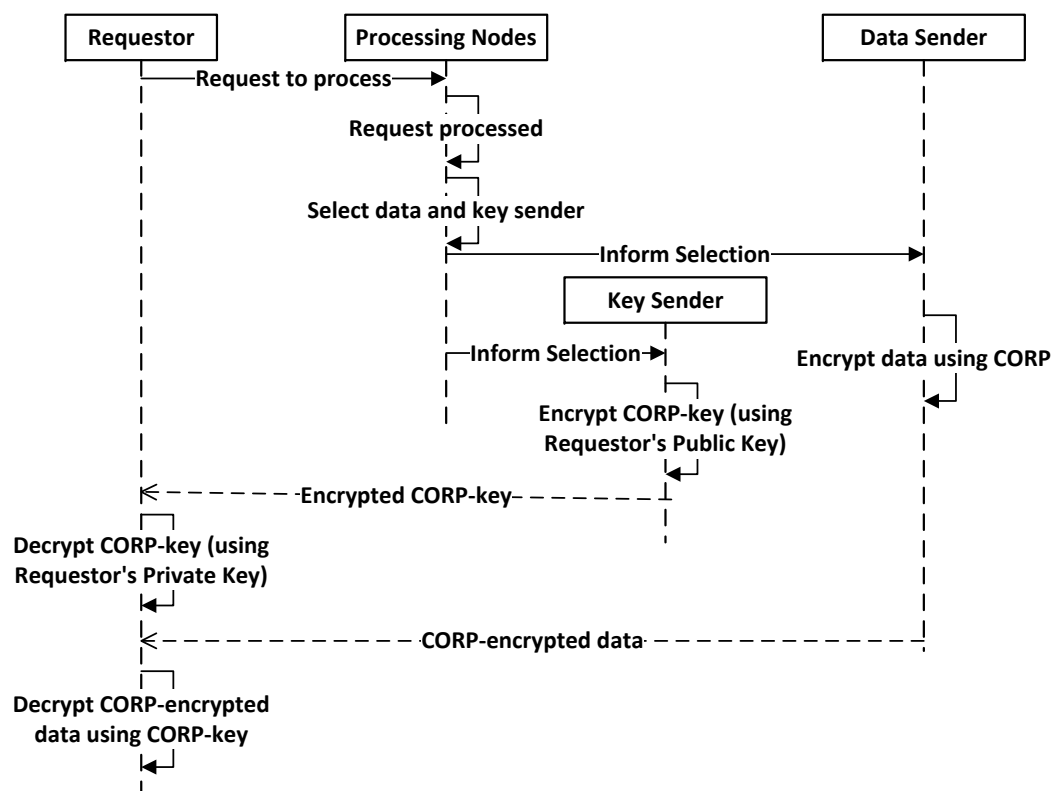


Figure C.3: Collaborative Processing using CORP

Upon receiving processing request from a requestor (end-user or another system), the processing nodes carry out the processing and then collectively decide and select one node as key-sender and one node as data-sender from themselves. Thus the requestor receives encrypted data from one node and the key to decrypt the data from another node. The integrity is thus maintained simply by using data and key from different sources.

C.4 Illustrative Example

Referring to the scenario in Figure C.1 on page 233, it is assumed that a processing is taking place for the component λ_1 . Since Node-A, Node-B and Node-D contains λ_1 , these three nodes will carry out the same processing. Once processing is done, the nodes will create checksum of the processed data. Since both nodes are working on the same data and carrying out exactly the same processing, their created checksum must match. At this point, both the nodes encrypt a pre-defined standard message by symmetric encryption, where the checksum is the key. The encrypted message is then multicast in the network by both Node-A, Node-B and Node-D among themselves without sharing the key, which is the checksum.

At this point, Node-A receives encrypted message from Node-B and Node-D, and vice versa. The nodes do not need to share the key since the checksum is used as key. Now, it is assumed that Node-A was able to successfully decrypt the message sent by Node-B, but the message from Node-D could not be decrypted. At this point, Node-A suspects an attempt to unauthorised data modification took place in Node-D and generates an exception flag. Node-A then multi-casts the exception flag to Node-B and Node-C. Thus the process of CORP algorithm ends with either of the two possible outcomes – exception flag or no exception flag.

C.5 Proof of Concept

Assumung the following:

$CORP$ = The Algorithm.

Number of nodes = $N_i \{i = 1, 2, \dots, N\}$

$P(x)$ = Successful decryption.

$Q(x)$ = Unsuccessful Decryption.

$N_iP(x)$ = Successful decryption in node N_i .

$N_iQ(x)$ = Unsuccessful Decryption in node N_i .

R = Exception Flag.

Thus, according to CORP algorithm, the successful decryption with no exception flag generated is denoted by the Equation C.1 below,

$$\forall N_i[N_iP(x) \rightarrow \neg R] \quad (\text{C.1})$$

According to the condition, even a single exception flag would mean some malicious act; and thus no inconsistency will exist only and only if there is no exception flag at all, this is denoted by Equation C.2,

$$\exists N_iP(x) \iff \forall N_iP(x) \neg N_iQ(x) \quad (\text{C.2})$$

Thus, the construct of the algorithm is in Equation C.3.

$$CORP = [\forall N_i[N_iP(x) \rightarrow \neg R]] \wedge [\exists N_iP(x) \iff \forall N_iP(x) \neg N_iQ(x)] \quad (\text{C.3})$$

C.6 Conclusion

At this stage, CORP is not fully optimized and there are some processing overheads associated with CORP. For example, involving all the nodes in processing will make the complexity of the algorithm grow exponentially. As part of future development, a sub algorithm as part of the bigger context of CORP is aimed to develop that will select a random number of nodes for processing instead of involving all nodes.

CORP is a research in progress as part of future enhancement of the security model. Ki-Ngā-Kōpuku may incorporate CORP to add data security on top of its current features. However, CORP is not constrained to be used only by Ki-Ngā-Kōpuku.

CORP can be used in any distributed computing setting. CORP introduces redundant processing which is an overhead, but the trade-off of the feasibility of such overhead depends on the sensitivity of the data. CORP may be a good fit in the contexts that deal with highly sensitive data that requires top-notch security assurance. Though CORP is part of future enhancement of Ki-Ngā-Kōpuku, the application of collaborative processing is not confined only to CC.