



PhD thesis

Exploring a resource allocation security protocol for secure service migration in commercial cloud environments

Karthick, G.

Full bibliographic citation: Karthick, G. 2022. Exploring a resource allocation security protocol for secure service migration in commercial cloud environments. PhD thesis Middlesex University

Year: 2022

Publisher: Middlesex University Research Repository

Available online: <https://repository.mdx.ac.uk/item/w0v22>

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant

(place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address: repository@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: <https://libguides.mdx.ac.uk/repository>

Exploring a Resource Allocation Security Protocol for Secure Service Migration in Commercial Cloud Environments



**Middlesex
University**

Gayathri Karthick

Department of Computer Science
School of Science & Technology
Middlesex University, London

A thesis submitted for the degree of

Doctor of Philosophy

March, 2022

I would like to dedicate this thesis to my loving parents, my husband, my sister, my daughter and in memory of my grandparents, my uncle and my brother!

Acknowledgements

First and foremost, thanks to God for giving me the strength to reach for the stars and chase my dreams. I would like to express my sincere appreciation and gratitude to the following people for helping me complete this thesis.

I am grateful to Dr Glenford Mapp, my director of studies and supervisor, for giving me the opportunity to work under him. We are always surprised about his interest in research, innovative ideas, support to achieve my dreams, provided guidance at every stage of my research. His advice and support were always helpful to do this thesis successfully. He gave me the freedom to work on various research projects and to explore innovative ideas.

I would also like to thank my other academic supervisors, Dr Florian Kammüller and Dr Mahdi Aiash for their encouragement and support during my studies. Their reviews, comments, and observations always gave me a much better understanding of my research topic.

I would like to express my heartfelt thanks to Terri Demetriou, the Research Degrees Administration team, and the School of Science and Technology for providing me with administrative support and all the required facilities and arrangements for doing my research.

The journey would have not been possible without my husband, Mr Karthick Kathirvel and my little cutie pie, Yathika Karthick for providing me with constant support, love, and encouragement during my study and life. I extend my thanks to my loving parents, my in-laws, my sister Mrs Geethu, Mr Udhay, little niece Dhakshita, my extended family and relatives for their love, happiness on my success and support throughout my life. I also dedicate this thesis to my friends who have always been a significant source of support in making my journey here a memorable and pleasurable one.

Abstract

Recently, there has been a significant increase in the popularity of cloud computing systems that offer Cloud services such as Networks, Servers, Storage, Applications, and other available on-demand resources or pay-as-you-go systems with different speeds and Qualities of Service. These cloud computing environments share resources by providing virtualization techniques that enable a single user to access various Cloud Services. Thus, cloud users have access to an infinite computing resource, allowing them to increase or decrease their resource consumption capacity as needed. However, an increasing number of Commercial Cloud Services are available in the marketplace from a wide range of Cloud Service Providers (CSPs). As a result, most CSPs must deal with dynamic resource allocation, in which mobile services migrate from one cloud environment to another to provide heterogeneous resources based on user requirements. A new service framework has been proposed by Sardis about how services can be migrated in Cloud Infrastructure. However, it does not address security and privacy issues in the migration process. Furthermore, there is still a lack of heuristic algorithms that can check requested and available resources to allocate and deallocate before the secure migration begins. The advent of Virtual machine technology, for example, VMware, and container technology, such as Docker, LXD, and Unikernels has made the migration of services possible. As Cloud services, such as Vehicular Cloud, are now being increasingly offered in highly mobile environments, Y-Comm, a new framework for building future mobile systems, has developed proactive handover to support the mobile user. Though there are many mechanisms in place to provide support for mobile services, one way of addressing the challenges arising because of this emerging application is to move the computing resources closer to the end-users and find how much computing resources should be allocated to meet the performance requirements/demands. This work addresses the above challenges by proposing the development of resource allocation security protocols for secure service migration that allow the safe transfer of servers and

monitoring of the capacity of requested resources to different Cloud environments. In this thesis, we propose a Resource Allocation Security Protocol for secure service migration that allows resources to be allocated efficiently is analyzed. In our research, we use two different formal modelling and verification techniques to verify an abstract protocol and validate the security properties such as secrecy, authentication, and key exchange for secure service migration. The new protocol has been verified in AVISPA and ProVerif formal verifier and is being implemented in a new Service Management Framework Prototype to securely manage and allocate resources in Commercial Cloud Environments. And then, a Capability-Based Secure Service Protocol (SSP) was developed to ensure that capability-based service protocol proves secrecy, authentication, and authorization, and that it can be applied to any service. A basic prototype was then developed to test these ideas using a block storage system known as the Network Memory Service. This service was used as the backend of a FUSE filesystem. The results show that this approach can be safely implemented and should perform well in real environments.

Contents

Contents	v
List of Figures	xi
List of Tables	xiii
Nomenclature	xiii
1 Introduction	1
1.1 An Overview of Commercial Cloud Systems	1
1.2 An overview of Cloud services	2
1.2.1 Cloud Advertisement in Commercial Environments	3
1.3 Vehicular Ad-hoc Networks	3
1.4 Mobile Edge computing	4
1.5 The evolution of mobile services	4
1.6 Resource allocation and security for the Mobile service environment	5
1.7 Secure protocol solution approach	5
1.7.1 Research Aims and Objectives	6
1.7.2 Research Question	6
1.7.3 Thesis Outline	7
1.7.3.1 List of Publications	8
2 Related work	10
2.1 Literature Review	10
2.1.1 Brief Introduction	10
2.2 Supporting Highly mobile Environments	10
2.2.1 Y-Comm Reference Framework	11
2.2.2 Cloud advertisement in Commercial Environment	12
2.3 Service Oriented Architecture (SOA)	12
2.4 Sardis Framework	12
2.5 Mobile Edge computing	14

2.6	Container technology	14
2.7	Resource allocation applied in Highly Mobile environment in Cloud	15
2.8	Research Gap	16
2.9	Chapter Summary	16
3	Research Methodology	17
3.1	Introduction	17
3.2	Service Migration by containers	17
3.2.1	Investigating different migration mechanisms	18
3.2.1.1	KVM	18
3.2.1.2	Docker	18
3.2.1.3	LXD CRIU	18
3.2.1.4	Unikernels KVM	18
3.2.2	Kubernetes Model	18
3.3	VANET Clouds	19
3.4	Experimental Testbed	20
3.5	Use cases	23
3.5.1	Fuse File System	23
3.5.2	Network Memory Server	23
3.6	Formal Methods approach	24
3.6.1	Formal Verification Method or Symbolic Models	24
3.7	Model checker tools	25
3.8	AVISPA tool	25
3.8.1	Architecture of AVISPA tool	26
3.9	ProVerif Tool	28
3.9.1	Architecture of ProVerif tool	28
3.9.2	Secrecy Formalization	29
3.9.3	Authentication Formalization	29
3.9.4	Comparison of AVISPA and ProVerif tools	29
3.10	Chapter Summary	30
4	Resource Allocation Algorithm (RAS) for Service Migration	31
4.1	Brief Introduction	31
4.2	Resource Allocation Algorithm (RAS) for Service Migration	31
4.3	RAS Server in detail	32
4.4	Resources in general	33
4.4.1	CPUs	33
4.4.2	Memory	34
4.4.3	Networking requirements	34
4.4.4	Storage	34
4.5	PSEUDO CODE for RAS	35

4.5.1	Advertising Cloud Formulation	35
4.5.1.1	General Notations	35
4.5.2	Receiving Servers Formulation	36
4.5.2.1	General Notations	36
4.5.3	Resource Allocation Server	37
4.5.3.1	General Notations	37
4.6	Chapter Summary	38
5	Introduction to RASP Protocol	39
5.1	Brief Introduction	39
5.2	RASP protocol – An Overview	39
5.2.1	The Server	40
5.2.2	Cloud Facilities	40
5.2.3	The Registry	41
5.2.4	Nonces (N) and Timestamps (T)	41
5.2.5	General Notations	41
5.3	Algorithm1 of first approach	42
5.4	RASP_Algorithm_V1	43
5.4.1	Stage 1: Advertisement	43
5.4.2	Stage 2: Authentication of SA and CB as well as migration request and response	43
5.4.3	Stage3: Migration transfer	44
5.4.3.1	Key Observations	44
5.4.4	Stage4: Update of New service location to the Registry	44
5.5	Evaluation of the First Attempt	46
5.5.1	HLPSL Specification	46
5.5.2	OFMC and ATSE	47
5.6	Rasp_Algorithm_V2.Second approach	48
5.6.1	Using Symmetric key	49
5.6.1.1	General Notations	50
5.6.2	RASP for Migration between server on Cloud CA to Cloud CB	50
5.6.3	The RASP Protocol in detail	51
5.6.4	Stages in detail	52
5.6.4.1	Stage 1	52
5.6.4.2	Stage 2	53
5.6.4.3	Stage 3	53
5.6.4.4	Stage 4	54
5.7	Modelling the Protocol by using ProVerif	54
5.7.1	Part1: Declarations	54

5.7.2	Part1a: Modelling Constructor and Destructor for Crypto primitives	55
5.7.2.1	Symmetric Encryption	56
5.7.2.2	Asymmetric key Encryption	57
5.7.2.3	Session key encryption	57
5.7.2.4	Signatures	57
5.7.3	Part2: Process macros	57
5.7.4	Part3: main processes	58
5.7.4.1	Evaluation of the Second Attempt	60
5.7.5	ProVerif Results	60
5.7.5.1	Nonces are secured and not derived by the attacker	60
5.7.5.2	The session key is not derived by the attacker . .	60
5.7.5.3	Private keys of SA & CB are not derived by the attacker	61
5.7.5.4	Authentication SA to CB and CB to SA is true .	61
5.7.6	Query attacker ()	62
5.8	Chapter Summary	62
6	New mechanisms to provide more security in Service Environments	63
6.1	Brief Introduction	63
6.2	Capabilities	63
6.2.1	Capability structure	63
6.2.1.1	Type Field (8 bits)	64
6.2.1.2	SYS Field (4 bits)	64
6.2.1.3	Property Field (12 bits)	65
6.2.1.4	Object ID (72 bits)	65
6.2.1.5	Random Bit Field (16 bits)	65
6.2.1.6	Hash Field (16 bits)	65
6.2.2	Rules for Capabilities	65
6.3	New Service Management Framework	66
6.4	SSP Protocol - An overview	68
6.4.0.1	General Notations.SSP Algorithm	71
6.5	ProVerif results	73
6.5.1	Queries for Private keys	73
6.5.1.1	Queries for nonces	74
6.5.1.2	Queries for Symmetric key	74
6.5.1.3	Queries for authentication of Server event and CMS event	75
6.6	Chapter Summary	75

7	Implementation	76
7.1	Brief Introduction	76
7.2	Basic capability System Library (BCSL)	76
7.3	Filesystem structure	77
7.3.1	Data blocks	78
7.3.2	The structure of CAP FILE	78
7.3.3	inode	79
7.3.4	RPC	80
7.4	Normal NMS operation	81
7.4.1	Fuxfs server	81
7.4.2	Fuxfs client	82
7.4.3	Make Fuxfs Multithreaded	82
7.5	SMF	83
7.5.1	Register Service	83
7.5.2	Register User	83
7.5.3	Register Device	83
7.5.4	Add Server	83
7.5.5	Request Service	84
7.5.6	Migrate Service	84
7.6	Testing and Evaluation	84
7.6.1	Docker in detail	84
7.6.2	Docker Hub	84
7.6.3	SMF - FUSE as a Service	84
7.6.4	Steps for FUSE Server Migration in Docker Mechanism	85
7.7	FUSE Server Migration in Docker - Results	85
7.7.1	Source code compilation in Command Prompt	85
7.7.2	Docker build - Fuse server	85
7.7.3	Create private repository in Docker Hub	86
7.7.4	Push a Fuse container image to Docker Hub	87
7.7.5	Run image from Docker Hub	89
7.7.6	Starting the Service Management Framework	90
7.7.7	Fuxfs server start up	90
7.7.8	Fuxfs client start up	91
7.7.9	Making the Super user	91
7.7.10	Make user	91
7.7.11	Make device	92
7.7.12	Make service	93
7.7.13	Add Server	94
7.7.14	Request service	94
7.7.15	Migrate service	94
7.7.16	Migrate service using Docker	96

CONTENTS

7.7.17 Migrate service - Container Running Status	97
7.7.18 Final results of SMF	97
7.8 Chapter summary	99
8 Conclusion and Future Work	100
8.1 Contribution of the Thesis	100
8.2 Contributions to the research	101
8.3 Contribution to the Field	102
8.4 Conclusion and Future Work	102
Bibliography	103
Appendix	109
.1 Docker installation	109

List of Figures

1.1	Cloud Layers	3
2.1	YComm Architecture	11
2.2	Service Migration Framework	13
3.1	VANET Clouds	19
3.2	VANET Clouds - FUSE and NMS	20
3.3	Full Coverage and Overlapping Map for A41, Watford Way, Hendon, London	22
3.4	Fuse Architecture with NMS	24
3.5	Architecture of AVISPA tool	26
3.6	Architecture of ProVerif tool	29
4.1	Resource Allocation Table	33
5.1	Migration from Cloud A to Cloud B	40
5.2	Migration from Cloud A to Cloud B	46
5.3	OFMC: Cloud A to Cloud B	47
5.4	ATSE: Cloud A to Cloud B	47
5.5	Migration from Cloud CA to Cloud CB	49
5.6	Second Attempt Migration between SA to CB	52
5.7	ProVerif - Declarations	55
5.8	ProVerif - Constructor and Destructor for Crypto primitives	56
5.9	Server SB processes	58
5.10	Cloud CB processes	59
5.11	Registry server	59
5.12	Main Process Macros	60
5.13	Second Attempt results using Proverif	61
6.1	Capability Structure	64
6.2	Capability Structure	64
6.3	Effect of introducing SMF in the client-server environment	67

LIST OF FIGURES

6.4	Service Migration Prototype	67
6.5	Secure Service Protocol	69
6.6	SSP protocols Results	73
7.1	Basic capability System Library (BCSL)	77
7.2	Normal NMS operation	81
7.3	Fuxfs Multithread	82
7.4	Fuse Server - Build	86
7.5	Docker image for Fuse Server	86
7.6	Docker Hub - Private repository Created	87
7.7	Docker Hub - Private repository	87
7.8	Fuse Container pushed to Docker hub	88
7.9	Fuse Container pushed to Docker hub	88
7.10	Run image from Docker Hub	89
7.11	SMF Execution	90
7.12	Fuxfs server start up	91
7.13	Fuxfs client start up	92
7.14	Making superuser	92
7.15	Make user	93
7.16	Make device	93
7.17	Make service	94
7.18	Add Server	95
7.19	Request service	95
7.20	Migrate service	96
7.21	Migrate service	96
7.22	Migrate NMS service using Docker	97
7.23	Docker Status: Fuse Container is running	98
7.24	SMF, FUSE Client and Server and Docker Status: Fuse Container is running	98

List of Tables

5.1	General Notions	42
5.2	General Notations	50
5.3	Query attacker of results	62
6.1	General Notations	72
6.2	QUERY ATTACKER() RESULTS	74

Chapter 1

Introduction

1.1 An Overview of Commercial Cloud Systems

Cloud computing is an emerging technology and along with mobile services, is rapidly becoming the next Internet-based enterprise platform. These cloud computing environments provide on-demand network access to a shared pool of configurable computing resources, (e.g. Networks, Servers, Storage, Applications), and any other available resources that can be rapidly provisioned with minimal management effort or extra provider interaction [Cloud \(2011\)](#). Clouds can be divided into public (e.g. Gmail, Amazon Elastic Compute Cloud (EC2), IBM's Blue Cloud, Google AppEngine, and Microsoft Windows Azure Services Platform), private (e.g., a company or school owns the resources and only members of that entity can have access to it) and hybrid deployments. Clouds can provide services in three different ways such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In cloud computing environments, there are two members: data owners or cloud users, and Cloud Service Providers (CSPs). On one hand, data owners are migrating their data to Cloud Storage platforms without buying any physical storage devices. Many companies find that it is the best way to manage the expenditure instead of buying, installing, maintaining computational resources. This approach gives many benefits to large enterprises and to individual users because it reduces the cost of storage services and allows these services to be managed in a more dynamic way. As a result, it has become a highly cost-effective way of delivering services to end-users. On the other hand, this has made Cloud Services popular among service providers who wish to offer processing and storage resources for various purposes. An increasing number of Commercial Cloud Services offered by several CSPs are available in the market and they hold massive computing resources in their Cloud servers/data centres and provide heterogeneous types of resources (Processors, Memory, and Network, etc.) at different prices and with the differ-

ent speeds and Qualities of Services (QoS). For example, the top Cloud Service Providers including Google, Microsoft, Amazon, and Rackspace actively advertising their Cloud Services. Additionally, these Providers will actively advertise their Cloud resources to services that will use these advertisements to dynamically migrate their servers to these Clouds. Cloud computing is, therefore, facilitating the migration of data and services. These services are called mobile services and will be used to support mobile users as they move around. These services can migrate between Clouds and thus reduce the latency between the service and mobile users, hence maintaining a good QoS for users. Additionally, the Cloud services are built using virtualization techniques as explored in the development of VMware and Citrix Systems. This thesis focuses more on commercial Cloud services rather than Private Clouds, to make our efforts useful to organizations.

1.2 An overview of Cloud services

The Cloud system offers a vast range of options, from the basics of computer storage, memory to powerful technologies such as Internet of Things (IoT) and Artificial Intelligence (AI) as well as standard web applications over the Internet. There are three types of Cloud services paradigms available [Zhang et al. \(2010\)](#).

The most widely known service is Infrastructure as a Service, where Cloud providers offer infrastructures such as data centre frameworks, virtual servers, storage, and networking. This is an advantage for start-up companies or existing organisations to run the company without the need to buy physical hardware for servers, etc.

The second type is Platform as a Service. It supplies the tools and technologies that allow applications to be built on top of virtual servers, networks, and storage. This service includes applications, database management, operating systems, and development tools.

The third type, Software as a service, is the most widely known and used on a day-to-day basis, such as web-based apps, backup storage, project management tools. SaaS therefore provide the entire suite of hardware and software resources to run businesses and enterprises using Cloud Services.

Clouds can provide the type of services in three different ways which are presented with examples in [Figure 1.1](#) below.

Public cloud: As the name implies, a public Cloud implementation makes its resources available to access via Internet. These clouds are open to everyone who requests the services which are offered (e.g. Gmail, Examples of public



Figure 1.1: Cloud Layers

clouds include Amazon Elastic Compute Cloud (EC2), IBM's Blue Cloud, Google AppEngine and Microsoft Windows Azure Services Platform).

Private cloud: It belongs to an organization and is only accessible to that organization for internal purposes. We refer to a private implementation as a community Cloud if it is used by numerous organisations with the same goal (e.g. A company or school owns the resources and only members of that entity have access to it).

Hybrid Cloud: Hybrid Cloud is a combination of the private and public Cloud approaches. As a result, while some data processing or storage is done on a private Cloud, the majority is done on a public Cloud.

1.2.1 Cloud Advertisement in Commercial Environments

Cloud computing services are becoming more popular and hence their services are actively advertised to other services. This is being implemented using beacon dissemination-based stations that will be deployed in busy places to send advertisements for selling services and products to users with mobile devices (e.g. Apple iBeacon).

1.3 Vehicular Ad-hoc Networks

Vehicular Ad-hoc Networks (VANETs) was first proposed and introduced in 2001 as car-to-car ad hoc mobile communication and networking applications, in which networks can be built, and information can be transmitted among cars. VANET is an example of a highly mobile environment and hence it will be investigated

for this research. Additionally, VANETS introduce the concept of beaconing. This can support the Cloud service marketplace as the first point for users to find out about different services. Due to the increase in the number of cloud advertisements, it is difficult to find the right advertisement within the usual timescale [Hussain et al. \(2012\)](#).

1.4 Mobile Edge computing

Mobile Cloud Computing integrates cloud computing with the mobile environment. For example, video streaming companies (e.g. Netflix, YouTube) and Cloud gaming (e.g. Google stadia) require low latency and high bandwidth, which must be guaranteed as users move around to ensure a better QoS for end-users. Support for seamless connectivity in highly mobile environments is now required as traditional reactive handover techniques have been found to be inadequate because of high speeds. The Y-Comm Framework [Mapp et al. \(2007\)](#) is one of the modern architectures designed to build future mobile networks by integrating communications, mobility, QoS, and security. The emergence of new networks such as VANETs needs services to be closer to the mobile user as they move around. Mobile Edge Computing (MEC) introduces a pristine environment that allows services to be moved closer to the user, which is from a centralized cloud to the edge of the network.

1.5 The evolution of mobile services

Service providers provide a pool of computing services to satisfy the computing needs of multiple users via the Internet. The pooled resources are distributed across many users based on their requests. For example, using virtualization techniques, such as VMware, Container technologies such as LXD, Unikernels, and Docker [Pentyala \(2017\)](#), orchestration technologies for containers such as Kubernetes, checkpoint/Restore in User-space (CRIU), proper resources allocation techniques and mathematical algorithms, these Cloud Services are supporting a large base of users with diverse needs. Now, what is therefore needed are proper resource allocation mechanisms to support mobile services. Furthermore, a service-oriented architecture that supports service migration in highly mobile environments by using multifunctional edge clouds, which are used to move servers closer to the location, is needed to migrate the services. It is important to develop a new service-oriented architecture to maintain useful resource management and secure service migration. Sardis proposed a reference model for service migration [Sardis et al. \(2013\)](#). However, this model doesn't cover security and privacy, which is the main concern hindering the large-scale deployment of mobile services.

It is crucial to provide a secure service to all users in a network and therefore, a new security framework must be looked at in detail. This research uses a concept of capabilities, and shows how a capability-based approach could be adopted to provide Authentication, Authorization, and Accounting is also called “AAA” for mobile users, applications, and devices.

1.6 Resource allocation and security for the Mobile service environment

Even though these Cloud Services have already played a vital role in the computing world, it suffers from several security concerns, challenges in resource allocation, and threat issues that can inhibit the functioning of entire Cloud systems and services. In particular, it is essential to ensure that servers do not end up being hosted on unsafe Cloud systems that can affect mobile services, and Clouds do not end up hosting malicious servers that can damage Cloud infrastructure [Karthick et al. \(2017\)](#). Hence, there is a big challenge on the Cloud resource management and security components in mobile clients. However, these challenges about Cloud Services are truly an opportunity to further enhance the Cloud to make it more useful to migrate their services to safe Cloud servers. Additionally, in these environments, resources must be quickly allocated and de-allocated as users move around [Paranthaman et al. \(2019\)](#). Hence, proper resource allocation management must be considered a key enabling mechanism to allow seamless connectivity and to provide QoS in highly mobile environments.

1.7 Secure protocol solution approach

In order to develop a viable solution, techniques of verification must be applied to prevent the process and a sequence of operations that ensures protection. The Formal Methods approach is the process of verifying system behaviour on modelling the security protocols and then defining the expected security properties of security protocols using formal semantics, assertions, and applying mathematics to the problem. It helps find ambiguity, inconsistency, incompleteness in the system specification and to enforce the desired behaviour of the system. This thesis considers resource allocation for service migration in Cloud environments that helps reduce operational costs, to build a widely distributed development and deployment team in organizations to access applications via the Internet, increase scalability, performance and expand the business geographically. Therefore, developing a Resource Allocation Security Framework for highly mobile systems would be the best option to achieve a realistic model which can be applied to

most service systems.

1.7.1 Research Aims and Objectives

This research addresses this issue in the context of the Service-Oriented Framework described by [Sardis et al. \(2013\)](#). Service migration has been proposed for many environments and is increasingly being used in Cloud infrastructure. Edge computing using Clouds introduced to reduce delays has been growing rapidly and enables support for mobile services in highly mobile environments such as Vehicular Ad-hoc Network (VANETs). Most of the work focused on achieving fairness through techniques like edge computing, service-oriented framework, and Y-Comm Architecture [Mapp et al. \(2007\)](#). Though all these mechanisms are promising developments, it is necessary to consider resource allocation and secure Cloud service migration in order to develop a new system for highly mobile environments.

- The project's technical objectives are to develop a resource allocation protocol that allows the migration of valid services to safe Cloud environments. This will allow services to be safely migrated closer to the users as they move around. These suggested mechanisms will be verified by AVISPA and the ProVerif tool.
- It is to incorporate new Capabilities and mechanisms for providing Authentication, Authorization, and Accounting(AAA) to authenticate the servers and the Cloud Systems.
- The next step is to develop a new Service Management Framework(SMF) that can use the RASP protocol to manage the migration or replication of the services in Cloud systems.
- All the above mechanisms will be incorporated with the RASP protocol to form a Secure Service Protocol(SSP) that can be used to manage any service.
- To develop a simple prototype of the system using a Basic Capability System Library (BCSL) and Docker with the Network Memory server as a service for migration.

1.7.2 Research Question

This thesis, therefore, looks at secure service migration and resource allocation in future networks. The key research question that must be addressed in this thesis is: ***“How can a Resource Allocation Security Framework be used***

for secure service migration to protect against attacks and be implemented in VANET cloud environments?”

This research question will be pursued by investigating the following issues.

1. How do we design a simple and fast resource allocation system that takes into account the resources available and the resources being used to allow the movement of services in a highly mobile environment?
2. What type of secure resource allocation protocol should be developed to securely migrate services from one Cloud system to another given available resources? How should we show this security protocol is correct using cryptographic model checkers such as the AVISPA tool and ProVerif tool?
3. What new mechanisms are to be introduced to provide authentication, authorization, and accounting (AAA) such that the system will also be secure from a user and device perspective?
4. In order to test these mechanisms, we will build a basic prototype that shows how these mechanisms can be implemented.

1.7.3 Thesis Outline

- Chapter 1: provides an introduction of Cloud computing, Cloud services, resource allocation, research questions, as well as research aims and objectives.
- Chapter 2: the related work presents background information of the existing solutions and approaches produced by researchers, scientists, and groups on Cloud computing environments and research analysis for resource allocation, attacks in service migration.
- Chapter 3: details the methods, algorithms, tools, and approaches used for conducting this research.
- Chapter 4: details the resource allocation algorithm.
- Chapter 5: Formal modelling and verification to verify an abstract protocol using AVISPA and ProVerif tools are presented in this chapter.
- Chapter 6: discusses the new mechanisms such as Capabilities, Service Management Framework and SSP protocol. SSP protocol verified using the ProVerif tool.
- Chapter 7: details the implementation of a basic prototype. Fuse/NMS is a use case which will be used in Docker container for service migration.

-
- Chapter 8: concludes this thesis with a summary of the research, contributions of this thesis, and directions for future work, to ensure continual improvement in our research.

1.7.3.1 List of Publications

The content of this document is part of on-going research, some of which was presented in the following publications:

1. Gayathri Karthick, Glenford Mapp, Florian Kammuehler, and Mahdi Aiash. 2017. Exploring a Security Protocol for Secure Service Migration in Commercial Cloud Environments. In Proceedings of International Conference on Internet of Things, Data and Cloud Computing, Cambridge, United Kingdom, March 2017 (ICC'17), 7 pages. DOI: <http://dx.doi.org/10.1145/3018896.3056795> [Karthick et al. \(2017\)](#).
2. Gayathri Karthick, Glenford Mapp, Florian Kammuehler, and Mahdi Aiash. 2017. Formalization and analysis of a Resource Allocation Security Protocol for Secure Service Migration, 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) [Karthick et al. \(2018\)](#).
3. Gayathri Karthick, Glenford Mapp, Florian Kammuehler, and Mahdi Aiash. 2019. Exploring a Secure Service Migration in Commercial Cloud Environments - The 26th Workshop on Automated Reasoning (ARW 2019), Middlesex University, London [Gayathri Karthick et al. \(2019\)](#).
4. Jose Guillermo Ramirez Gil, Onyekachukwu Augustine Ezenwigbo, Gayathri Karthick, Dr. Glenford Mapp, Dr. Ramona Trestian. A New Service Management Framework for Vehicular Networks. ICIN 23rd Conference on Innovation in CLOUDs, Internet and Networks(ICIN2020) [Ramirez et al. \(2020\)](#).
5. Onyekachukwu Augustine Ezenwigbo, Jose Guillermo Ramirez Gil, Gayathri Karthick, Dr. Glenford Mapp, Dr. Ramona Trestian - Providing Reliable Network Storage in Highly Mobile Environments: Challenges and possible Solutions. ICC2020 -IEEE International Conference on Communications [Ezenwigbo et al. \(2020\)](#).
6. Karthick, G, Mapp, G, Kammuehler, F, Aiash, M. Modelling and verifying a resource allocation algorithm for secure service migration for commercial cloud systems. Computational Intelligence. 2021; 1– 18 [Karthick et al. \(2021\)](#).

-
7. Vithanwattana, Nattaruedee, Karthick, Gayathri, Mapp, Glenford E. and George, Carlisle (2021) Exploring a new security framework for future healthcare systems. In: IEEE Global Communications Conference, 07-11 Dec 2021, Madrid, Spain [Hybrid: In-Person and Virtual] [Vithanwattana et al. \(2021b\)](#).
 8. Nattaruedee Vithanwattana^{1*}, Gayathri Karthick¹, Glanford Mapp¹, Carlisle George¹ and Ann Samuels²-Securing Future Healthcare Environments in a post-COVID-19 world: Moving from Frameworks to Prototypes, Springer Nature 2021 - submitted status.

Chapter 2

Related work

2.1 Literature Review

2.1.1 Brief Introduction

This chapter is broken down into three parts. The first part focuses on the technical background, which introduces us to numerous technologies designed to achieve seamless communication. Secondly, it details the related work of researchers who investigated the resource allocation applied in a highly mobile environment in the Cloud. Finally, the third part describes the research gap which will be the focus of this research effort.

2.2 Supporting Highly mobile Environments

In the computing world, many systems are associated with different machines that are geographically distributed. In the last few years, we have seen a dramatic improvement in Cloud Computing that provides resources as a service. It is because of the emergence of IOT devices, Big Data (e.g. flight signals), and social networks such as Facebook and Twitter, that produce a large amount of data that needs to be stored and migrated successfully to safe Cloud environments. Several research efforts were carried out looking at the migration, handover, storage and web applications for highly mobile environments but few addressed the issue of secure service migration in Cloud Systems.

Sardis's framework [Sardis et al. \(2013\)](#) and [Huang et al. \(2016\)](#) clearly show that researchers are interested in service management and hosting services on Edge Cloud servers. However, these efforts did not include looking at security in service migration and a new security framework as highlighted in Service Management Framework [Ramirez et al. \(2020\)](#). This chapter presents an extensive

analysis of various research efforts that investigated resources in Cloud, Y-Comm Framework, Service Management Framework, and resource allocation and secure service migration applied in highly mobile environments.

2.2.1 Y-Comm Reference Framework

In this new environment, mobile users will demand to be always connected using heterogeneous networking. Mobile devices will, therefore, have several wireless interfaces including Wi-Fi, LTE, 5G, satellite and Ultra-Wideband interfaces. These networks will seamlessly work together using vertical handover techniques. The OSI model is no longer sufficient for dealing with connectivity in the modern era of mobile users and devices. Hence, Y-Comm is an architecture (Figure 2.1(b)) that has been designed to build future mobile networks by integrating communications, mobility, QoS and security. It accomplishes this by dividing the Future Internet into two frameworks: Core and Peripheral frameworks as shown in Figure 2.1(b) Mapp et al. (2007).

Cloud Systems should be able to allocate their resources without delay in order to ensure a sustainable QoS for mobile users. Indeed, it has been suggested that we need a modern architecture which provides communication, mobility and QoS and security. Hence, the Y-Comm framework is considered one of the key concepts of enhancing QoS in Vehicular Clouds. The Internet is currently evolving. Instead of large, global, but individually-managed networks, a core network is being deployed which is fast and getting faster with peripheral wireless networks situated at its edges. A Core Endpoint is an entity which is at the edge of the core network and is used to connect different types of wireless systems as shown in Figure 2.1(a). Vertical handover techniques are used to maintain the connection between the MN and the core network via the Core Endpoint as the user moves around.

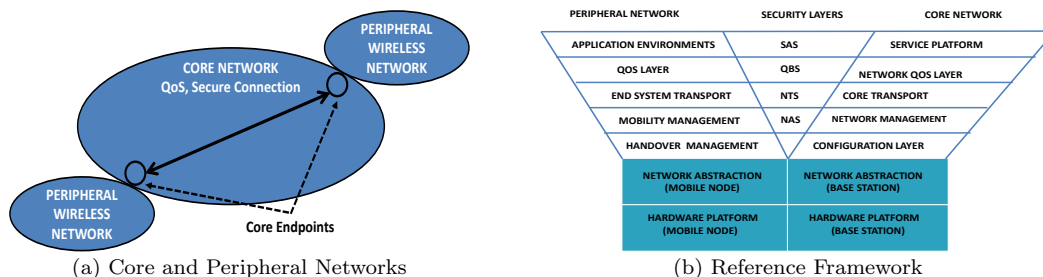


Figure 2.1: YComm Architecture

The researchers of Y-Comm have made major contributions in the areas of

proactive handover as well as introducing new concepts in security such as Targeted Security Models.

A more detailed description of Y-Comm framework is given [Mapp et al. \(2016\)](#). However, it should be noted that Y-Comm supports a layer for mobile services called the Service Platform Layer in the Core Network. This layer allows services to be installed and managed coherently in future networks including support for mobile services.

2.2.2 Cloud advertisement in Commercial Environment

The authors [Ullah et al. \(2016\)](#) attempted to address the development of a secure commercial advertising scheme for Vehicular networks. They proposed a scheme name called “Business discovery (BUY)” that presents the concept of the beaconing market using a Vehicular network. The authors [Xiao et al. \(2016\)](#) have observed, that among various vehicular applications, the most promising strategies involve the dissemination of commercial advertisements via car to RSU and car to car communication.

2.3 Service Oriented Architecture (SOA)

Cloud-based services and Service-Oriented Networks can improve QoS and load balancing on a global scale even further. In the Service Oriented Architecture (SOA) model, heterogeneous services are invoked via their interfaces to create more complex services such as support for Virtualization [Erl \(2007\)](#). Cloud computing is an example of a SOA application, where many components and services that are running on various parts of the infrastructure come together to provide a client with a whole business solution. Network resources running on underlying infrastructure can be virtualized with the use of the same service-oriented paradigm, thereby separating the infrastructure from the network services. In [Shashwat and Kumar \(2017\)](#), the authors proposed a model called the “service identification model for service-oriented architecture” which helped to reduce the response time of the services and find the right service within the specified time. Service identification will be easy if we follow distributed-based functionalities for our service.

2.4 Sardis Framework

A new service-oriented framework for mobile services was proposed by Sardis, to support the automatic migration of services based on the amount of traffic they generate and on what QoS their clients are receiving. To achieve this, a service

must be aware of its own geographical location and the locations of its clients. Additionally, it will also require access to a global map of Cloud locations to be used as a reference for moving itself as close as possible to its clients. To provide a complete set of mechanisms to enable mobile services, it was necessary to develop a new service-oriented architecture that allows services to be managed, copied, or migrated to support mobile users. The system should also provide algorithms that incorporate traffic management and the QoS requirements of the flow. This new framework was proposed in [Sardis et al. \(2013\)](#) and has six layers as shown in [Figure 2.2](#), Service framework is briefly described below

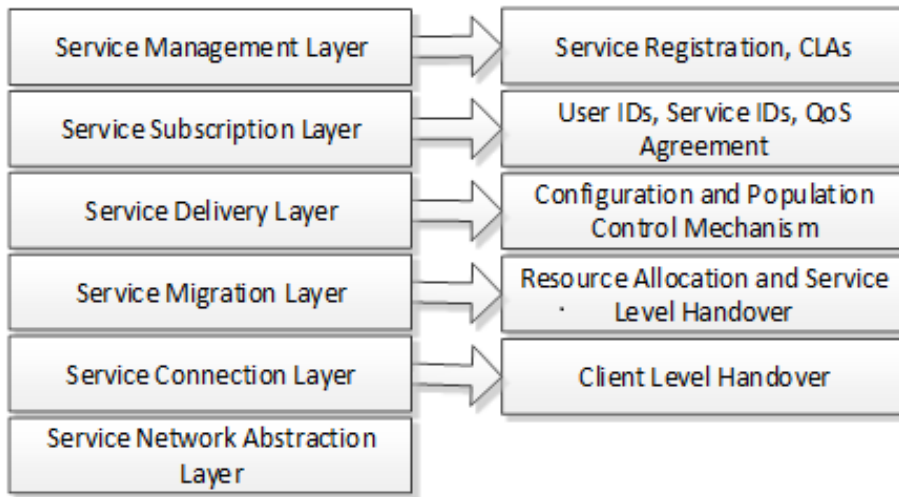


Figure 2.2: Service Migration Framework

This framework is a powerful framework that can be used by all services to migrate servers from one Cloud to another Cloud. The work of Sardis showed that to migrate a service, it is necessary to compare the time taken to migrate the service with the amount of time the user will be in the region concerned. Hence, the mobility model of the user must be considered. Sardis used a simple queuing model to represent user mobility in mobile networks. Using this service framework, Sardis demonstrated several mechanisms to ensure the migration of services including algorithms to determine when it is beneficial to migrate the services looking at the time it would take to move the service to the new network. Additionally, an analytical model was developed to analyze the increase in latency as the mobile user moved away from the service, hence it was possible to satisfy both requirements. However, the security aspects of migrating services using Cloud Interoperability mechanisms fell outside the scope of his research. There are several other aspects of the proposed system that present potential security weaknesses. The main security concern is that of ensuring that performance

data for an individual client and service are exchanged in a way that prevents impersonation or tampering.

2.5 Mobile Edge computing

Mobile cloud computing (MCC) provides a seamless connection in mobile applications and devices by integrating Cloud computing into the mobile environment [Li et al. \(2016\)](#). However, the growing number of Cloud services (e.g. OneDrive, Zoom Cloud meetings), and the realization of new services (AI, IoT, smart wearable devices such as fitness trackers, smart clothing, smartwatches, etc.) and advanced services (self-driving, Augmented reality, virtual reality) include high network load. Mobile Edge computing (MEC) is a new network architecture that moves the computing of traffic and services from a centralized Cloud to the network's edge, bringing it closer to the mobile user. It's now an upward paradigm that offers endless possibilities and performance metrics such as high availability, distributed, low latency, direct access to real-time network information, and high bandwidth approach for user workloads. MEC improves delay-constrained offloading in Cloud-enabled vehicular networks. The authors in [Zhang et al. \(2016\)](#) proposed a vehicular offloading framework in a Cloud-based MEC environment. They investigated the computation offloading mechanisms and could check the latency and resource limitations of MEC servers. This work enabled the proposal of a computation and resource allocation. The method described by [Kikuchi et al. \(2017\)](#) is a MEC-based VM migration scheme in which VM migration is performed to reduce congestion at the edge of the network. They solved two QoS issues: the congestion of wireless access networks and congestion of computing resources at the edge based on network TCP throughput by considering various network scenarios that increase network TCP throughput. Therefore, to achieve a fast and secure service migration, we need the services at the edge of the network between Edge Clouds.

2.6 Container technology

This section describes service migration technologies that existed from the old to the latest. Virtualization is a computing software that allows users to use multiple OS, programs and deploy apps on one physical computer [Gao and Tang \(2013\)](#). System virtual machines provide substitutes for a real machine and provide the functionality needed to execute the entire operating system. A hypervisor uses native execution to share and manage hardware, allowing for multiple environments which are isolated from one another, but still exist in the same physical

machine. Process virtual machines are designed to execute computer programs in a platform-independent environment.

The entire machine is virtualized down to the hardware layer, and the container virtualizes only the software layer above the operating system level. Many container technology mechanisms have made migration possible, such as Docker, KVM, and Universals.

2.7 Resource allocation applied in Highly Mobile environment in Cloud

As discussed in [Tordsson et al. \(2012\)](#), the most used service models of Clouds are data centres, virtualization Clouds, network resources, storage, CPUs, firewalls, and load balancers. The major Cloud Service Providers are Amazon Web Services (AWS), Google, and Salesforce and simultaneously, the active larger IT firms are Microsoft, IBM, and Dell. Therefore, many Commercial Cloud Services are public Cloud computing services. The growing challenge is how Commercial Cloud Services can efficiently allocate resources to meet the requirement of QoS. In [Paranthaman et al. \(2019\)](#), the authors proposed a new method to support proactive resource allocation for future networks such as Vehicular Ad-hoc Networks. As network resources are usually shared between many users, resource management must be a key part of communication systems to ensure that, applications and servers receive their required QoS. In [Paranthaman et al. \(2019\)](#), the authors investigated how Time Before Vertical Handover (TBVH) and Network Dwell Time (NDT) for a mobile node in any given networking topology can also be used to aid the proactive management of resources by analyzing the contention between mobile users for communication channels in wireless networks. The research determined the times when different nodes will need to acquire and release resources due to mobility. Though contention analysis has been used to analyze systems such as wired and wireless networks, this work greatly extends this approach to look at mobile heterogeneous environments where different cellular technologies may be used. The approach is particularly suited to heterogeneous systems where many networks may be operating at the same time in the local area. Finally, the proposed approach only uses the mobility of the node and coverage of the network to determine the key input parameters. These parameters can be determined in great detail for any networking technology and so this approach can be integrated with other analytical techniques. Authors in [Paranthaman et al. \(2017\)](#) investigated a new methodology to support proactive resource allocation in highly mobile networks for emerging future networks such as 5G by allowing base stations to calculate the probability of contention based on the demand for network resources. Additionally, the proposed approach used Markov

chains to model the contention and showed the results based on system performance in terms of throughput and mean response time. However, the authors did not consider service migration. In [Hamdy et al. \(2017\)](#) the authors proposed many resource allocation strategies and their challenges in which a Cloud system can be defined as any mechanism that aims to guarantee that physical or virtual resources are assigned correctly to Cloud users. For example, Linear Scheduling Methods, Virtual machines, Nature Inspired Optimization Methods, Gossip Protocol Based Methods, and Priority Based Methods. To efficiently allocate resources, Cloud computing deals with various resource allocation techniques [Beloglazov et al. \(2012\)](#) and the dynamic allocation model [Bhaavan et al. \(2014\)](#). However, no strategy talked above about security in highly mobile environments. The main challenge is how secure resource allocation can be possible in highly mobile environments.

2.8 Research Gap

Industries are increasingly embracing Cloud computing and the fast-developing Internet. Simultaneously, the explosive growth of information has become a big challenge to network security as discussed in [Dutta and Hammad \(2020\)](#). Meanwhile, these security concerns also highlight the need for security protocols and algorithms to ensure secure communication between the parties as highlighted in [Blanchet et al. \(2016\)](#). Typically, the goal of the service providers is to generate revenue with minimum investment and to maximize resource usage. Resources, such as CPU, memory, storage, network, and I/O must be allocated in an appropriate amount according to the capacity of Cloud servers in service migration. Hence, resource allocation algorithms must be considered a key enabling mechanism to allow seamless connectivity and to provide QoS in highly mobile environments. This thesis addresses these issues by proposing a Resource Allocation Security Framework for commercial Cloud environments.

2.9 Chapter Summary

This chapter examined the concepts around resource allocation in networks and service including the Y-Comm Reference Framework supporting heterogeneous networks, Cloud advertisement in Commercial Environment, and Sardis' reference framework for service migration, MEC is a new service model that requires service migration to provide better QoS to clients. Furthermore, it looked at other research concerning resource allocation in the highly mobile environment. The research gap identified was the need to provide secure service migration from one to another Cloud server using secure resource algorithm techniques.

Chapter 3

Research Methodology

3.1 Introduction

This research has the aim to develop a resource allocation security protocol for secure service migration in commercial cloud environments. It is essential to discuss the tools and techniques we used in our research. This chapter includes containerization, formal methods and the Middlesex VANET Testbed which are used in this research.

3.2 Service Migration by containers

Service migration has been proposed for many environments and is increasingly being used in Cloud environments that support virtualization. Virtualization is forming a virtualized version of a service like a server or a network to create multiple execution environments. This is possible because the virtual machine paradigm allows entire virtual machines to be migrated. It improves Cloud agility, performance, and scalability among Cloud users. However, virtual machine migration can be expensive as the entire virtual machine has to be moved. The emergence of containers as a lightweight technology to virtualize applications has been leading to successful systems in service migration and particularly in managing applications in the Cloud. The collection of nodes is called “Clusters.” Often, management using container-enabled clusters is essential and the orchestration of construction and deployment have been driven by the need for faster service or higher QoS. There are many container technology mechanisms, which are gaining in prominence, that have made migration possible such as Docker in which containers house several services.

3.2.1 Investigating different migration mechanisms

3.2.1.1 KVM

Kernel based virtual machine (KVM) is an open source technology and needs their own kernel instance to run while containers share the same kernel or allows the kernel to act as an hypervisor. However, these approaches assume that the communication architecture does not support server migration, making it difficult to apply these mechanisms in Wide Area Networks (WANs). Whenever a user uses remote resources there is always a chance that sensitive data may fall into wrong hands which involves the packaging of applications. There is an interest in this area and hence I would like to explore a Resource allocation security framework for secure service migration helping organizations securely move services and applications to the Cloud.

3.2.1.2 Docker

Docker is an open-source containerization system. It has a small and lightweight execution environment which the packaging of applications, operating systems, source code with its dependences, links, libraries, and versions required to run that code in any environment.

3.2.1.3 LXD CRIU

This supports for live migration in containers. Lxd depends on LXC and it enables virtualization with its own process and network space.

3.2.1.4 Unikernels KVM

Unikernels KVM in which the operating system is bounded and customized to run a single main application is the next emerging specimen of this genre. From a management point-of-view, it makes server migration simpler.

3.2.2 Kubernetes Model

Kubernetes is a portable, widely used, open-source orchestration system [Medel et al. \(2016\)](#) that provides APIs for automating the deployment, scaling and management of containerized applications. It was created by Google and originally coded in the Go programming language and is now maintained by the Cloud Native Computing Foundation. It is compatible with several container tools, including the Docker mechanism. Although Kubernetes has many applications, it is limited by a lack of QoS, the use of a basic resource algorithm, and a lack of in-depth analytical modelling.

3.3 VANET Clouds

Vehicular Ad-hoc Networks (VANETs) are a key part of Intelligent Transport Systems(ITS) to provide efficient and safe communication between Vehicles and Infrastructure (V2I) using Road Side Units (RSUs) in Figure 3.1. This includes data exchange between high-speed vehicles and between the vehicles and the roadside infrastructure. It provides low latency and high bandwidth therefore, it is an ideal environment in which to test our work. In Figure 3.2 shows three

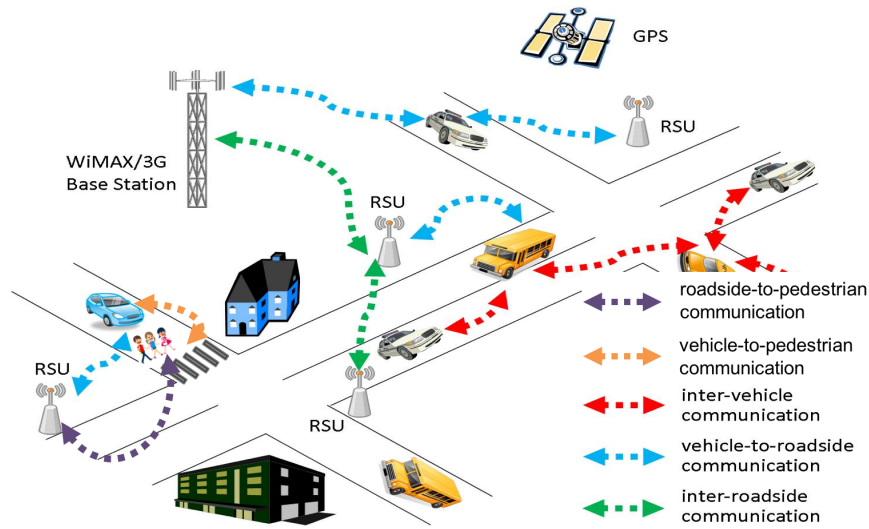


Figure 3.1: VANET Clouds

wireless access points which are called Core Endpoint can be replaced as the RSU referring back to the live experimental test-bed conducted [Paranthaman et al. \(2016\)](#). Core Endpoints are connected to the core network; therefore the services can run from the core network and also from the Core Endpoint. With the help of Mobile Edge Clouds, we are able to run the service at the Core Endpoint; this is to ensure low latency. When a mobile user using the VANET as a means of communication, and then moves from Core Endpoint A to B while using an internet service such as video streaming, the services will be migrated from Core Endpoint A to B. In other words, to be able to move services from point A to B or to C, a new service layer must be dedicated to the migration of services. This layer takes into account of mobility, security and policy aspects as well as the throughput, bandwidth and latency. The main reason why we are looking into VANET is because it is an example of a highly mobile environment and hence it will be investigated for this research. This helps us to address the research question of migrating services from one server to another in cloud environments.

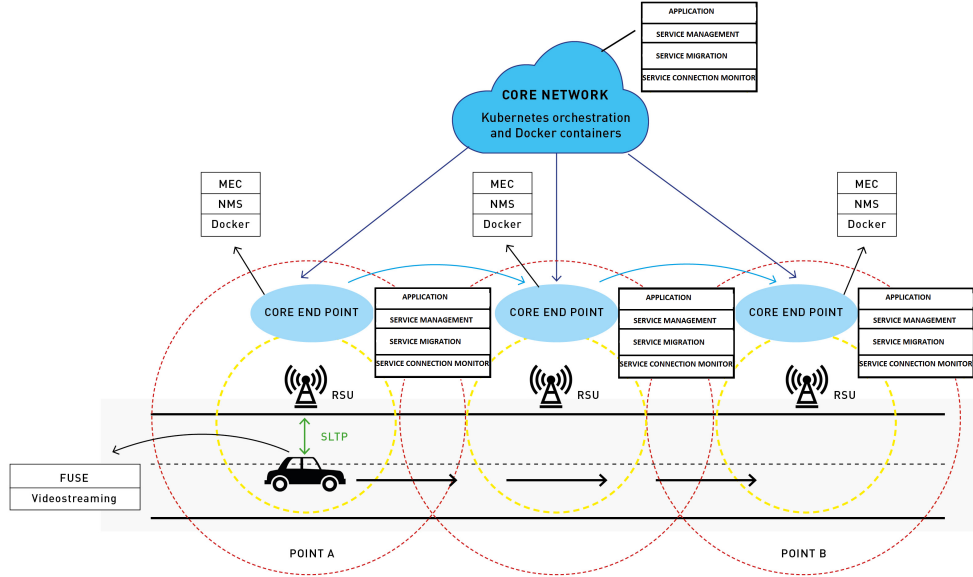


Figure 3.2: VANET Clouds - FUSE and NMS

3.4 Experimental Testbed

The distribution of Connected and Autonomous Vehicles (CAVs) would change the environment we live in. In general, Connected Vehicles permit us to build an ITS by enabling strong connectivity among vehicles and the transport infrastructure. This is referred to as Cooperative-ITS (C-ITS). The deployment of C-ITS will result in better traffic and road management, fewer accidents, shorter journey times, better collision avoidance mechanisms and increased efficiency to manage major disasters. For better understanding nowadays, the building of new technology is a necessity; the use of testbeds and applications will give us a better understanding of this new era. Middlesex University and the Department for Transport (DfT) have built a Connected Vehicle Testbed that uses ITS-G5 (VANET) technology. This section provides the details of the real experimental VANET testbed. The Connected Vehicle Testbed was built by Middlesex University and the DfT using ITS-G5 technology. The testbed was built on the Hendon Campus in London and alongside the surrounding roads and then extends to the A41 (Watford Way) behind the campus. Four RSUs which were deployed in the MDX buildings were backhauled using the university's gigabit Ethernet network and the three RSUs deployed along the A41 were backhauled using LTE with a secure Virtual Private Network (VPN) tunnel service provided by Mobius Networks. They are now fully operational and trials have been held to fully understand the technology and concerns around its wide-scale deployment as well as communication dynamics to attain seamless communication for this environ-

ment. The individual coverage ranges attained by the RSUs located along the A41 road and RSUs located on each MDX building are shown in the coverage map using various coloured dots in Figure 3.3. The Hendon Campus and surrounding roads were covered by the first four (1-4) RSUs - the area covered was about 0.7 miles/1.1 km. The other (5-7) RSUs covered the A41 as the coverage was from the between the entrance of the Great Northern Way (above the brown line) to Hendon Central Tube Station (underneath the blue line) – with a distance of 2 miles/ 3.2 kms. As such, the total coverage of the testbed was approximately 2.7 miles/ 4.31 kms.

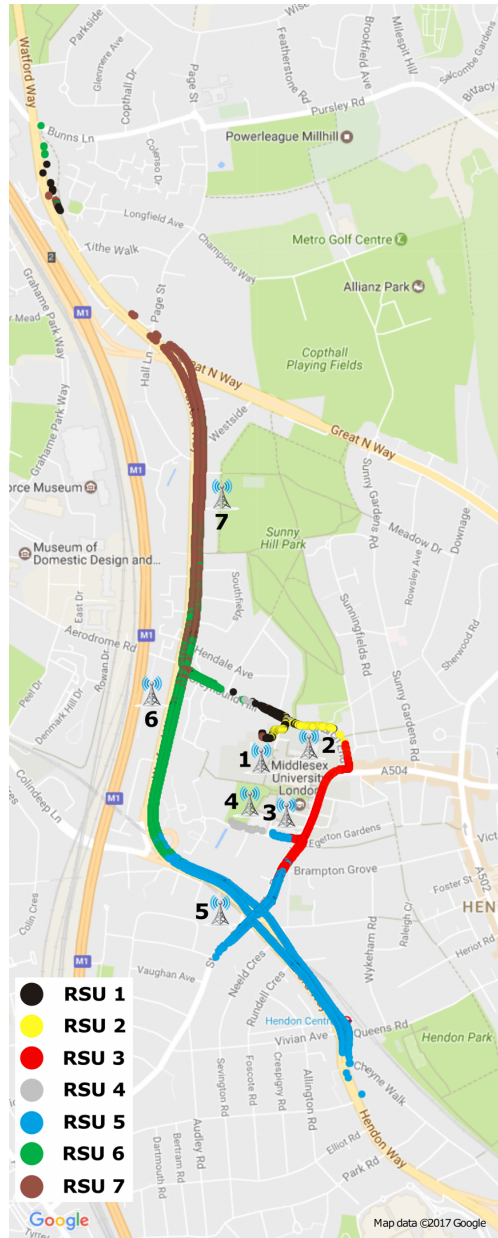


Figure 3.3: Full Coverage and Overlapping Map for A41, Watford Way, Hendon, London

3.5 Use cases

3.5.1 Fuse File System

File systems play a crucial part in every operating system. It is a place where users keep their files. The organization of the file system also plays an important role in helping the user find files. There are many file systems that have been developed by different programmers. Every system has its own advantages and disadvantages. It is vital for a user to select the most suitable file system. Having a suitable and appropriate file system enables the computer system to operate at higher efficiency. Additionally, a file system allows the user to attach special attributes to the file such as the ownership of the file and permissions over the file. The most common file system that are used by end users is NTFS which is short for New Technology File System. Some file systems have special features, some offer better reliability and robustness and some provide quicker read and write speeds. However, there are some file systems do not run in the kernel; instead, they run in user-space. This provides better flexibility as the kernel is complicated. Hence, in my research, we show that by implementing File system in User space (FUSE) with the Network Memory Server which is discussed below, will provide more flexibility to potential NMS users [Vangoor et al. \(2017\)](#).

FUSE is a software interface that provides a bridge from the user space to the kernel. This allows the file system to be placed user space and outside of the kernel space. According to Layton et al, the illustration in Figure 3.4 describes a file system named hello is compiled and being executed. When hello is executed, the FUSE mounts the “test” file system in the directory “/tmp/fuse”. Here, the user can store his/her data using this system. All the data will be stored in the user space directory “/tmp/fuse”. Then the user executes an “ls -l” as shown in the top left of fig, this command goes through the glibc to the VFS in the kernel. The VFS then goes to the FUSE module. The FUSE module will contact the hello filesystem through the glibc and libfuse (which is the FUSE library in user space) and asks for the result of the command. The result will then be returned back to the FUSE module and passed through VFS and finally to the “ls -l” command.

3.5.2 Network Memory Server

The Network Memory Server (NMS) is an example of a simple, stateless service; it stores blocks of data from clients in its memory (RAM). Clients can create, read, write and delete blocks of data. The NMS is primarily a storage platform for mobile users. In this thesis, we will explore the NMS as an example of a mobile service as discussed in [Chang et al. \(2017\)](#). In this research we will use the NMS

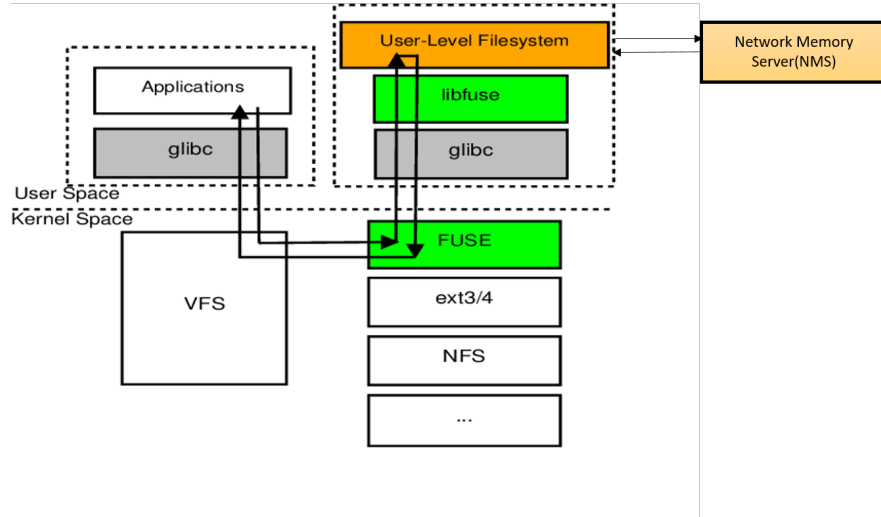


Figure 3.4: Fuse Architecture with NMS

as a back network store for the a FUSE file system on the mobile node. Hence, we will build a platform in order to migrate the NMS using Docker. The applications that we will be looking at running on the NMS are, Streaming Applications, Web services, and Vehicular services. The FUSE architecture with the NMS is shown in Figure 3.4.

3.6 Formal Methods approach

3.6.1 Formal Verification Method or Symbolic Models

Industries are increasingly embracing Cloud computing and the fast-developing Internet. Simultaneously, the explosive growth of information has become a big challenge to network security. Meanwhile, these security concerns also highlight the need for security protocols can be used to prove their security properties or detect unpredictable attacks and discover new vulnerabilities. Therefore, security protocols must be defined rigorously in order to be trusted to have some degree of assurance that these protocols fulfil their goals. Security Protocols (SP) are known as “cryptographic protocols” when we add security mechanisms such as encryption, decryption, authentication, hash functions, digital signatures, session keys between them until they communicate confidentiality. For example, the TLS protocol was believed to be secure for 13 years while the Needham-Schroeder Public key protocol was believed to be secure for 17 years before Lowe discovered that it had a vulnerability attack. Therefore, in practical applications, security protocols may be unruly realize their security properties or unpredictable attack

and vulnerability still exist. As a result, the correctness analysis of security protocols has become an important job to detect attacks in cryptoprotocols [Chen et al. \(2016\)](#)). Additionally, the goals of security protocols are ensuring secrecy, authentication, integrity, anonymity, and establishing session keys between entities. However, in hostile environments, hostile agents are referred to as attackers, penetrators, or intruders that are deliberately trying to undermine the protocol.

The Formal Methods approach is often called the Dolev-Yao model due to Needham Schroeder and Dolev-Yao cryptographic primitives. Here, crypto primitives are black boxes and messages are terms, modelled by functions, symbols in an algebra of terms, possibly with equations, and facilitates automatic proofs. Here, the attacker has full control over all communications. In formal verification methods, the protocol is analyzed to say whether the security properties are met or not, or a reason is given why they are not met through an analysis attack trace. Hence, rigorous methods have been developed for verifying security protocols. The formal method approach is a more effective method to find attacks on security protocols. Some of the formal verification methods are abstract state machines, Theorem proving, model checkers [AR and Devane \(2010\)](#). Additionally, the process of this method verifies system behaviour on modelling the security protocols and then defines the expected security properties of security protocols using formal semantics, assertions, and applying mathematics to a problem. We typically also want to analyze these protocols, where the adversary can look at the resources, public keys, or compute the limited set of functions such as read or delete any messages.

3.7 Model checker tools

Model checking is a technique that relies on building a finite model of a system which is encoded into a modelling language and checking whether the desired property holds in the model or not by providing an attack graph [Clarke and Schlingloff \(2001\)](#). Various tools are available these days to verify the security protocols. In this report, two popular cryptographic verification tools namely AVISPA and ProVerif are used for analysis of the protocol.

3.8 AVISPA tool

AVISPA is an automated push button tool for the Automated validation of Internet security protocols and Applications(AVISPA). It makes use of High-level specification language and offers a modular, expressive formal language for expressing security protocols and features (HLPSL). This tool combines many backend tools that use a number of automatic analysis techniques, including attacks on the

input protocol and infinitely many abstraction-based sessions of verification. For example, IKEv2 ASW and Radius-SHA256.

3.8.1 Architecture of AVISPA tool

The architecture of the AVISPA tool is depicted in Figure 3.5. The HLPSL is an expressive, modular, role-based, formal language that combines several back-ends that carry out various automatic analysis methods. It also allows for the specification of protocols and their security features. We believe that no other tool offers the same level of scope and stability while still performing well and scaling easily. The AVISPA tool uses an automatic translator to convert the user defined protocol/problem into an intermediate format(IF) specification. This IF specification describes an infinite-state transition system that is readily applicable to formal analysis for states that represent attacks on the intended properties of the protocol [Armando et al. \(2005\)](#). The tool integrates with four back-end tools:

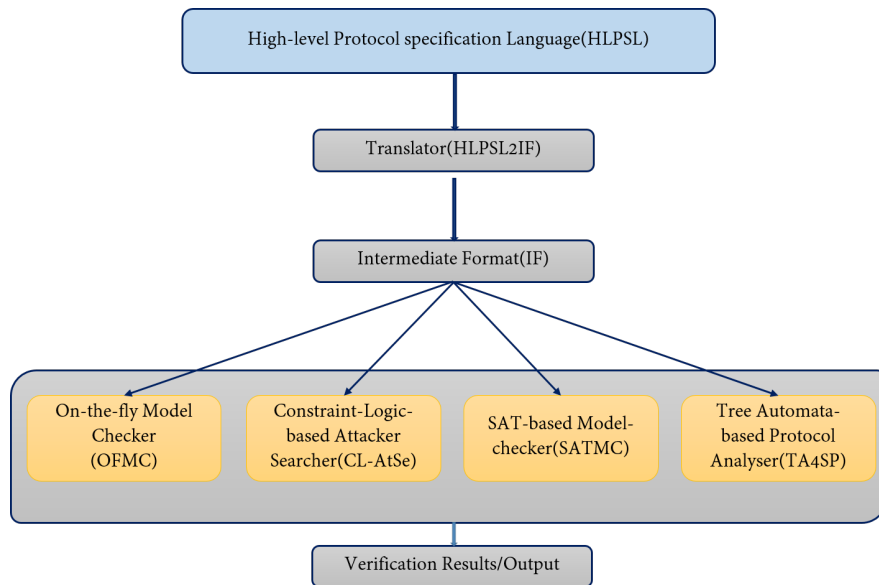


Figure 3.5: Architecture of AVISPA tool

- On the On-the-fly Model Checker(OFMC),
- the Constraint Logic-based Attack Searcher (CL-AtSe),
- SAT-based Model Checker (SATMC)
- Tree Automata-based Protocol Analyzer (TA4SP)

Each back end tool has its own options and components to define the formal verification [Hurtado Alegría et al. \(2014\)](#), [Armando et al. \(2005\)](#).

- **OFMC**: The OFMC performs protocol falsification and bounded verification for a number of sessions by exploring transition systems by an IF specification in a skill manner. In addition, it considers both typed and untyped models, supports algebraic properties of Crypt operators, and implements number of symbolic techniques such that they do not lose any attacks or introduce any new ones.
- **CL-AtSe**: The CL-AtSe performs protocol falsification and verification for a limited number of sessions. The protocol messages can be either typed (For example, integer) or untyped, handles XOR operator, and an extensions of handling algebraic properties. This tool, built in a modular way with some powerful simplification heuristics, performs several types of optimization thereby reducing interruptions, redundancies, inefficiencies and unnecessary branches in the protocol symbolic executions.
- **SATMC**: This tool considers the typed protocol method and performs protocol falsification as well as finite number of session verification. This tool can be used to generate a propositional formula that encodes a finite unrolling of the IF-described transition relation, the initial state, and the set of states that indicate a breach in the security properties in the protocol. The modern SAT solver is then fed the propositional formula, and any models it uncovers are converted back into attacks. This tool also supports for reducing redundancies .
- **TA4SP**: This tool performs infinite protocol verification by attempting to guess the intruder’s knowledge through the use of regular tree languages and rewriting. When it comes to a protocol’s secrecy properties, this tool can determine if it is flawed or whether it is secure for any number of sessions.

The four back-end tools of the AVISP implement different analysis techniques. There is communication between the tools, and you need to specify the protocol in HLPSL with the properties you want to verify, then invoke the back-end tool you require. The Dolev-Yao model specifies the exchange of messages over the network (DY model). After the execution process, the output describes the results of the protocol. AVISPA provides the similar set up like DY Model to test the security protocols in an environments as if they were tested in the real world. During the simulation step, the DY model simulates an adversarial model to identify any security weakness in the protocols. The output is listing the common format of Summary, Details of bounded number of sessions and the goals. The AVISPA model checker, a widely accepted tool, provides the results of whether the security

protocol is SAFE or UNSAFE. If the attacker can break the proposed security protocol, the protocol declared as "UNSAFE". On the other hand, the proposed protocol runs without any security breach, the protocol declared as "SAFE" [M \(2022\)](#).

3.9 ProVerif Tool

ProVerif performs automatic symbolic protocol verification by analysing cryptographic protocols. It supports a variety of equations, rewrite rules (Constructor and destructor), and cryptographic primitives. For infinite message spaces and an unbounded number of sessions, it can demonstrate a number of properties, including secrecy, authentication, and process equivalences. Horn clauses and a portion of the Pi calculus are the two types of input files that ProVerif accepts. It uses security attributes as input and does verification using a dialect, form, or extension of the applied pi-calculus or pi-calculus with cryptography. The applied pi calculus is a language for modelling security protocols and it is a language for studying concurrency and process interaction (e.g. Certified Email [Abadi and Blanchet \(2005\)](#)). ProVerif is capable of providing reachability properties, correspondence assertions and observational equivalences [Bruno Blanchet and Sylvestre \(2021\)](#). Cryptographic protocols are concurrent programs which interact by using Internet or public channels. Following the Dolev-Yao model, the attacker can see the channels which are controlled by cryptographic mechanisms.

3.9.1 Architecture of ProVerif tool

ProVerif has a large variety of protocol structures (event and the queries, private channels, rewrite rules, etc.) and modelling primitives used by cryptographic protocols (encryption, decryption, digital signatures, etc.). The ProVerif internal abstraction uses queries to verify the security properties and attempts to prove that there is a state in which the security properties are known to the attacker or not. If the results of the queries are "True", then the security property cannot be derived by the attacker. Additionally, it verifies the properties of the security protocol to prove secrecy (strong/weak), authentication and observational equivalence for an unbounded number of sessions using an unbounded message space. [Figure 3.6](#) depicts how to declare the syntax for specifications of protocol, process and results.

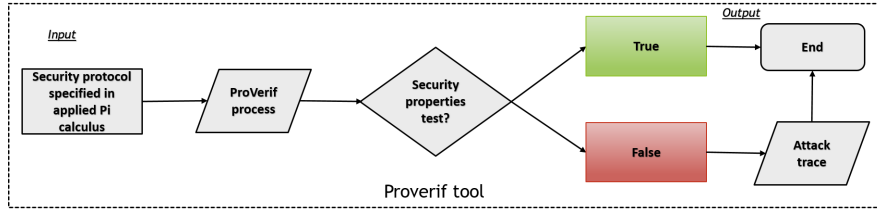


Figure 3.6: Architecture of ProVerif tool

3.9.2 Secrecy Formalization

In the ProVerif tool, to verify the secrecy of the term "Message" in the model, the following query function helps to find the confidentiality. Before the main procedure, the following query is added to the input file to test the confidentiality of the word "Message" in the model: `Query attacker(Message);`

3.9.3 Authentication Formalization

In protocols, the definition of the correspondence is used to capture the relationships between the events, which can be expressed as Statement 2 having been executed, then Statement 1 having been previously executed. Each of those statements includes parameters, which allows a relationship between the parameters of events to be studied. For an example from Automatic Cryptographic Protocol Verifier book [Bruno Blanchet and Sylvestre \(2021\)](#), the syntax to query a basic correspondence assertion is: `"query x1 : t1, . . . , xn : tn ; event (e(M1, . . . , Mj)) ==i event (N1, . . . , Nk) ."` The concept of "injective correspondence" assertions captures the one-to-one relationship and authentication between each occurrence of the event or the number of protocol runs performed by each entity if desired.

By creating queries that accurately test the security characteristics against an active attacker, ProVerif verifies the security properties to achieve security goals. ProVerif was successful in validating confidentiality, authenticity, key exchange, forward secrecy, and weak secrecy in the context of an active Dolev-Yao adversary.

3.9.4 Comparison of AVISPA and ProVerif tools

Using the AVISPA tool, we can show that the protocol specified runs without any security breach. However, in order to create a complete security framework for Cloud Systems, we also need to look at verification of the cryptographic protocols. Furthermore, AVISPA is sufficient for abstract protocols but ProVerif may help to better model the Cloud infrastructure, into the protocol thus allowing more extensive properties to be detailed and proved [Blanchet et al. \(2016\)](#). When the

number of executions of the protocol is not bounded, the problem is undecidable or it does not provide whether the proposed protocol runs with attacks or securely passed the events. Hence, there exists no automatic tool that always terminates and solves this problem. However, ProVerif uses an abstract presentation of protocols by Horn Clauses, and it retains relational information on messages which has better precision than AVISPA tree Automata(TA4SP) [Blanchet \(2013\)](#). For example, ProVerif allows input files for XOR or Exclusive-Or algebraic properties to verify Diffie-Hellman which OFMC and ATSE does not support [Lafourcade et al. \(2009\)](#). AVISPA tool installation is an image request/link request based installation which is sometimes unstable. ProVerif installations are standard OCaml based development [OCa \(2019\)](#) which makes downloading and installing the tool simpler. If we need help, they have a forum to give support.

Compared to AVISPA, ProVerif is a more expressive tool because it retains relational information on messages and produces results according to that. Hence, ProVerif allows us to better define what is being attacked and hence what needs to be protected. ProVerif also features efficient automatic reasoning tools and is therefore potentially able to verify specific properties such as Cloud resources [Aish et al. \(2012\)](#). In our second attempt, we used the ProVerif tool to fully verify the interaction for secure service migration including the security such as the services, nonces, private keys, session keys, signatures, encryption and decryption mechanisms. For example, ProVerif is used to verify security protocols in some fields such as Blue tooth security and E-voting systems.

3.10 Chapter Summary

This chapter gave a clear understanding of the need for security protocols, Formal Methods and how to model the security protocols by using formal verifier tools such as AVISPA and ProVerif for the verification of specification of protocols. To use a more realistic model, the abstract methods have to be tested in a real-time VANET environment. Here, we have explained Use cases of our research such as FUSE and NMS, therefore, the FUSE file system will be using NMS technology at the back end.

Chapter 4

Resource Allocation Algorithm (RAS) for Service Migration

4.1 Brief Introduction

Resource management is crucial for delivering excellent productivity in cloud infrastructures. In contrary to traditional infrastructures such as parallel and distributed systems, resource management in cloud infrastructures requires managing virtual machines (VMs), as all modern IaaS providers make extensive use of virtualization. This research aims to develop a resource allocation algorithm to check the capacity of the Cloud services before it migrates. In recent times, people have been moving to a solid-state drive (SSD) to access operating systems and programs.

4.2 Resource Allocation Algorithm (RAS) for Service Migration

Cloud computing delivers on-demand services and supports the current technologies such as the Internet of things, smart home, and visualizations. Resource allocation in Cloud is a challenging task for servers to migrate cloud services. Currently, there is no agreed resource allocation mechanism for migrating services in Cloud environments. Therefore, we propose a simple algorithm based on two critical components for each Cloud System.

1. One is total resources in terms of CPU, Storage, Network and Memory etc, which is fixed accordingly to the capacity of the system.
2. The second component is available resources which can be allocated to migrating services.

Each Cloud System advertises its total and available resources to mobile services. In order to efficiently migrate the service, the first step is to verify whether it is feasible to migrate based on the server requirements and the second step is to ensure that the service migration to the new Cloud System can be securely achieved. Once the server has verified that it is feasible to migrate the service, then RASP protocol is invoked to do the transfer. In the RASP protocol, the Registry also contains information about the capacity and available resources on the Cloud and thus can verify that the new Cloud System is a valid Cloud and has the resources to host the server. In addition, according to the RASP protocol the server running on the new Cloud, will inform the Registry that the service migration is completed and therefore the Registry will update its Resource Database for the previous and the new Cloud Systems.

4.3 RAS Server in detail

Each Cloud Service Provider can receive several requests for resources from their advertisements for building complex applications with different topologies and networking requirements.

The clouds advertise to various servers. In this scenario, we can consider that we have the first advertising cloud CA (1AC), the second advertising cloud CB(2AC), and the third advertising cloud CC(3AC), etc. The first advertising Cloud CA(1AC) broadcast its resources in terms of CPU, Memory, Network, and Storage to First Receiving Servers(1SA), Second Receiving Servers(2SA), and Third Receiving Servers (3SA), etc. The Cloud system advertises only available resources to the receiving servers. Let's assume 1SA migrates the services such as CPU, Memory, Network, and Storage to 2AC. This formulation is shown below. The Resource Allocation Server (RAS) is a trusted party which is mainly used for verifying the registration of the cloud and validating the registered cloud's capacity of the resources. It maintains the complete table of registered servers, registered clouds and their resources as shown in Figure 4.1. It has algorithmic operations to prove whether the cloud is valid or not and whether it has enough resources to give the requested resources.

As stated above, Cloud providers as $CP = \{ CA, CB, CC, CD, \dots Cn \}$. Each provider has a finite set of amount of resources to share and make available to the requested sources. Each Provider actively advertises its resources at each round, and executes our mathematical formulation (calculations 1, 2 and 3) to find the optimum sharing to server and accepts requests for resources from servers.

Resource Allocation Server (RAS)	First Advertising Cloud CA(1AC)	Second Advertising Cloud CB(2AC)	Third Advertising Cloud CC(3AC)	Advertising Cloud ...
First Receiving Servers(1SA)	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage
Second Receiving Servers(2SA)	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage
Third Receiving Servers(3SA)	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage
Receiving Servers ..	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage	CPU, Memory, Network and Storage

Figure 4.1: Resource Allocation Table

4.4 Resources in general

4.4.1 CPUs

CPU refers to the central processing unit acting as the computer's brain, allowing devices to perform operations, calculations, logical comparisons through applications. The speed of the CPU is calculated by how many instructions it can be handled per second. The AMD Opteron series and the Intel Itanium are CPUs used in servers and computers. ARM CPUs are smaller in size, less expensive, more efficient in handling clock speed that can reach 5.0 GHz, symmetric multiprocessing capabilities of up to 8 sockets/128 cores. ARM processors are Reduced Instruction Set Computer (RISC) processors and are used in mobile devices like smartphones and tablets. The Intel processor is more powerful, uses a set of RISC-like micro-instructions, clock speed, which can reach 5.0 GHz and symmetric multiprocessing functionalities of up to 4 sockets/28 cores can be used in Laptops, servers, and gaming. More powerful CPUs make the computer complete tasks faster but may also consume more energy. This updated version of CPUs can be used in AI and IoT.

4.4.2 Memory

Memory is a device or system used to store information temporarily or permanently in a computer or any hardware. Having a faster memory will give you better PC performance and faster processing speed than more memory. However, having more memory size available allow more processes to run simultaneously resulting in better overall performance. For example, we always set more memory sizes for VM instances. Basic Random-access memory (RAM) size is 8 GB, which is good for basic gaming, 16 GB is suitable for OS, 32 is an ideal size for storing and processing information. Additionally, Ram Cloud is super high-speed storage for applications and data centres.

4.4.3 Networking requirements

Providing Quality of Service (QoS) in networks is becoming an essential process because it helps manage network traffic, reduce packet loss, and lower network jitter and latency. Additionally, link reliability reveals the signal strength, and error detection helps find any impairments or mismatches with the senders and receivers' data. For example, services such as video on demand, online cloud meetings, streaming media and VoIP use Quality of service. The current network standards are 100 Mbps, 1Gbps. The higher speed network cards are available on the market, such as ASUS XG-C100C Pci-E X4 Card with Single RJ-45 Port – using 10Gbps networking. The missing key component in resource allocation of networks is QoS in terms of bandwidth, latency (delay), jitter, error detection and link reliability.

4.4.4 Storage

Network storage devices help computers connect through networks providing access to heterogeneous clients. Before that, we used a hard disk drive (HDD) that has a speed of 5ms to 10 milliseconds for storage and provides high storage capacity and low cost. People are moving to a solid-state drive (SSD) to access operating systems and programs in recent trends. SSD has a speed of 50-100 microseconds, faster than an HDD. Direct Memory Access (DMA) is an advanced technology in-network that accesses the host memory directly without CPU intervention. The extension of DMA is called Remote Direct Memory Access (RDMA), does the same job and has high speed and low latency, while transferring information from one device to another at the memory to memory level. Suppose we need more significant memory space, in that case, a new technology called Distributed Shared Memory (DSM) provides large virtual memory, allowing multiple processing elements to share the same location among several

processors. All this availability of resources means that there is a need to develop a simple and fast resource algorithm which will allow us to decide when is the best time to migrate a service to another Cloud platform.

4.5 PSEUDO CODE for RAS

In order to focus on this main idea, each cloud can have a set of resources due to its hardware, in addition each Cloud is uniquely identified by a Cloud ID and since they are Cloud services. As we know, a Commercial Cloud will have a number of resources which it actively advertises to servers. Modelling of resource demands, we must define what are the available resources of cloud.

4.5.1 Advertising Cloud Formulation

In Cloud formulation, let us indicate that Second Advertising Cloud (2AC) resources as any resources the total amount of resources allocated at 2AC cannot exceed Cloud's Max resources. (MAX_Cm). In stage 1 RASP protocol, 2AC is actively advertising its cloud-based services or free resources($Free_Cf$) such as CPU (Cfc), Memory (Cfm), Storage (Cfs) & Network(Cfn). 2AC has another set of variables called allocated(ALLOC) variables which tell it all the resources that have been already allocated: These values change as servers migrate to and from 2AC. Allocated Resource of CB($ALLOC_Ca$) such as CPU (Cac), Memory (Cam), Storage (Cas) & Networks(Can). Maximum Resource of Cloud (MAX_Cm) variables tell the maximum resource capacity of that Cloud such as CPU (Cmc), Memory (Cmm), Storage (Cms) & Networks(Cmn). At each round, each provider will use the mathematical formulation below, Step 1 from RASP protocol is as follows:

- **Stage 1** 1. $CB \rightarrow SA : Advc (CB, ResCB)$

4.5.1.1 General Notations

Maximum Resource of Cloud (MAX_Cm) = $\{Cmc, Cmm, Cms, Cmn\}$,
 Allocated Resource of Cloud ($ALLOC_Ca$) = $\{Cac, Cam, Cas, Can\}$ and
 Free Resource of Cloud ($FREE_Cf$) = $\{Cfc, Cfm, Cfs, Cfn\}$

Here, the letter "C" refers to the identity of the cloud CB, "Cf, Ca, Cm" refers free resources, allocated and maximum resources and c,m,s and n refers to no of CPUs, Memory, Storage and Networks respectively.

Advertising Cloud Formation in detail,

- **Step 1:** The resources CPU, Networks, Memory, Storage are members of resources(r).

Algorithm 1 Advertising Cloud Formation

- 1: Resources(r) \in {Cpu(c), Network(n), Memory(m), and Storage(s)}
 - 2: MAX_Cm (Cmr) = { Cmc, Cmm, Cms, Cmn }
 - 3: $ALLOC_Ca$ (Car) = { Cac, Cam, Cas, Can } &
 - 4: $FREE_Cf$ (Cfr) = { Cfc, Cfm, Cfs, Cfn }
 - 5: $Cfr \leftarrow Cfr \cup Resources(r)$
 - 6: If $Cfr \leftarrow (Cmr - Car) > 0$ then
 - 7: Send Adv of Cfr and wait for response;
 - 8: else $Cfr \leq \phi$ then
 - 9: wait for $Cfr > 0$;
 - 10: End if
-

- **Step 2-4:** 2AC's maximum, allocated and free resources are declared.
- **Step 5:** Inclusion of an element r to a set of free resources.
- **Step 6-7:** 2AC advertising it's resources with its identity 2AC.
- **Step 8-10:** If the above condition fails, 2AC needs to wait for the free resources until its greater than 0.

4.5.2 Receiving Servers Formulation

The server is represented here as First Receiving Servers(1SA) and it receives an advertisement request from 2AC and its free resources. A server that needs to use a Cloud has variables that detail its requirements. If free resources of *AdvertisingCloud2AC* { Cfc, Cfm, Cfs, Cfn } are larger than Receiving Server 1SA's($SAreqr$) requirements, it can forward the request to Registry (Resource Allocation Server(RAS)). If we assume that, requested resources are of lesser capacity than free resources of *AdvertisingCloud2AC* and all requested resources must be true, then only it sends the request to RAS.

- **Stage 2 2.** SA \rightarrow R : (SA, CB, ResCB)pkR

4.5.2.1 General Notations

Requested Resource of Server ($SAreqr$) = { Src, Srm, Sms, Srn } Here, the letter "S" refers to the identity of the Receiving Server, "reqr" refers requested resources and c,m,s and n refers to no of CPUs, Memory, Storage and Networks respectively, Receiving Servers Formation in detail,

- **Step 1:** The resources CPU, Networks, Memory, Storage are members of resources(r).

Algorithm 2 Receiving Servers Formation

- 1: Resources(r) \in {Cpu(c), Network(n), Memory(m), and Storage(s)}
 - 2: REQ_Resources 1SA ($SAreqr$) = { Src, Srm, Sms, Srn }
 - 3: $SAreqr \leftarrow SAreqr \cup \text{Req_Resources}$
 - 4: If $SAreqr > 0$ then
 - 5: Search for Resource Migration;
 - 6: If $SAreqr \leftarrow (Cf - SAreqr) > 0$ then
 - 7: Accept Cf and sends to RAS;
 - 8: Else ignore the request;
 - 9: End if
-

- **Step 2-3:** 1SA's Requested resources are declared. Inclusion of an element Req_Resources to a set of 1SA request resources.
- **Step 4-5:** If Requested resources are greater than 0, the cloud services needs to find the next best fit server.
- **Step 6-9:** If the free resources of 2AC subtracts Requested resources, which is greater than 0, then accept the advertisement and forward the request to RAS server to verify the validity of resources. Else ignore the advertisement.

4.5.3 Resource Allocation Server

RAS is a trusted party that is primarily used for confirming the registration of the cloud and certifying the registered Cloud's capacity of the resources. Note that the proposed algorithm will run at each provider involved in service migration to the requested servers. Once it receives the request from SA requested resources($SAreqr$), registry verifies the resources based on the below algorithm.

- **Stage 2 3.** $R \rightarrow SA: \text{sign}((CB, pkC, ResCB), pkS)$.

4.5.3.1 General Notations

- Resource Allocation Server (RAS),
- Registry(R).

If Registry receives the request from requested servers,
Server RAS Formation in detail,

- **Step 1-2:** RAS server verifies whether its a valid cloud and if Cloud resources are greater than the requested resources which is not equal to 0 or

Algorithm 3 RAS_CB

- 1: If $Cf = \text{Valid Cloud}$ and $RAS \leftarrow (Cf > SAreqr) \neq 0$ then
 - 2: Accept and sends an authenticated message;
 - 3: Else
 - 4: Reject the request;
 - 5: End if
-

not. If either condition false, then it rejects the request and updates 2AC that its an invalid server.

- **Step 2-5:** If the above conditions pass, then it sends an authenticated message to 2AC.

Algorithm 4 RAS_SA

- 1: If $SAreqr = \text{valid server}$ and $RAS \leftarrow (Cf > SAreqr) \neq 0$ then
 - 2: Accept and sends an authenticated message;
 - 3: Else
 - 4: Reject the request;
 - 5: End if
-

Server RAS Formation in detail,

- **Step 1-2:** RAS server verifies whether it is a valid server and if Cloud resources are greater than the requested resources which is not equal 0 or not. If either conditions are false, then it rejects the request and updates 1SA that its an invalid Cloud.
- **Step 2-5:** If the above conditions pass, then it sends an authenticated message to 1SA.

4.6 Chapter Summary

This chapter looked at a simple resource allocation algorithm to allow the migration of servers and services to different Cloud environments to support users as they move around. In highly mobile environments such as Vehicular environments, there is not much time in the network and, hence, we cannot run very complex resource allocation algorithms. That is why we developed a simple resource allocation algorithm. This research now focuses on a resource allocation security protocol for the safe migration of services between Cloud environments.

Chapter 5

Introduction to RASP Protocol

5.1 Brief Introduction

Service providers offer a pool of computer resources in order to meet the computing requirements of multiple users over the Internet. Several users receive a share of the pooled resources according to their requests. To improve the Quality of Service, providers collaborate on the resources from the pool to assign to each Cloud. Securing service migration in a cloud is one of the most challenging tasks; Cloud resources are vulnerable to abuse, intercept, or data loss. So, guaranteeing services enhance QoS and reduce the chances of losing service. Security protocols are becoming the main topic in the networking industry and verifying them has provided a significant development in the computer world. If traditional security mechanisms such as encryption and decryption remain relevant for Cloud infrastructure, they need more specific protection from new threats or attacks. To support secure migration in Commercial Clouds, we propose the Resource Allocation Security Protocol (RASP), an emerging standard for safe migration over Cloud infrastructure. This research will be performed with two methodologies, i.e., Formal modelling and Real-Time Test-bed experiment. We have shown two types of approaches. In our first approach, this RASP protocol was verified using the AVISPA tool and proved confidentiality and authentication between the participants. In the second approach, this protocol has been verified using the ProVerif tool and shown confidential, authentication between the parties and also secure session key migration.

5.2 RASP protocol – An Overview

This abstract protocol has three roles: Server SA on Cloud CA, Cloud CB, and Registry. The below Figure 5.1 shows the outline of RASP protocol.

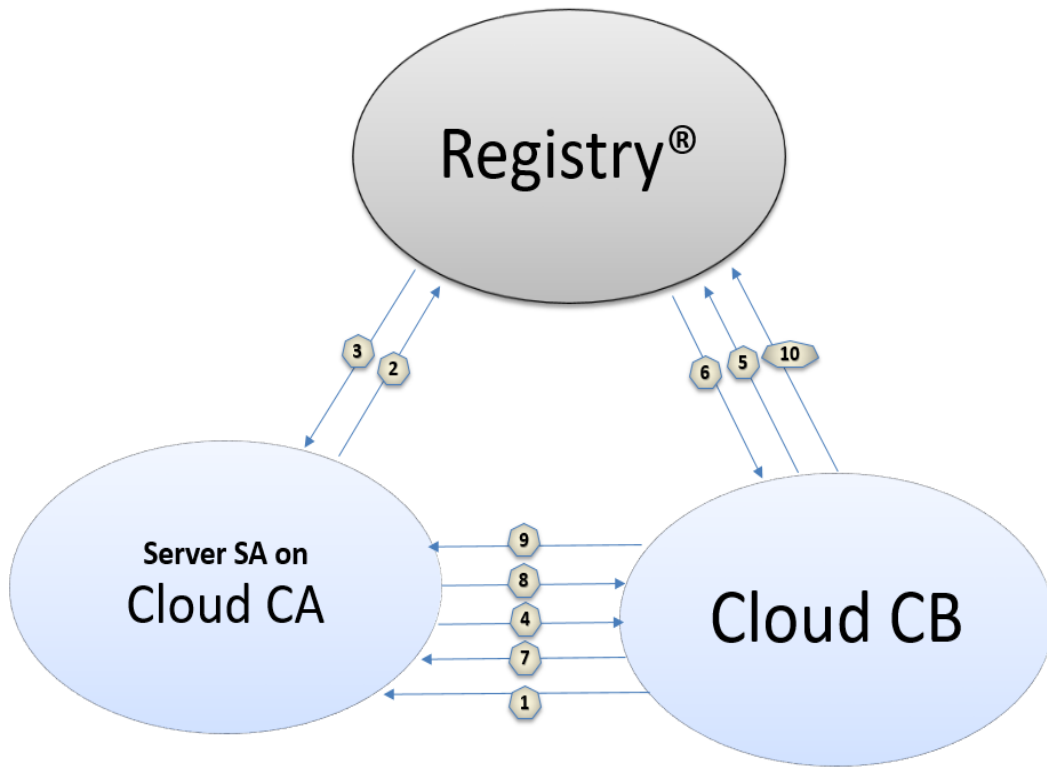


Figure 5.1: Migration from Cloud A to Cloud B

5.2.1 The Server

The Server (S) is represented by Server Identity (Server_ID), a type of service (TOS). A Public Key is denoted as PKS. Each server is identified by a unique server identity given to the server when it registers with Certificate Authority.

5.2.2 Cloud Facilities

Cloud Servers are represented as follows: CLOUD (CA) = Cloud_ID_CA, Types of Server is Cloud, a public key of Cloud CA and Available Resources. CLOUD (CB) has its unique identity Cloud_ID_CB, Types of Server is Cloud, a public key of Cloud CB, Requested Resources. A Cloud Identity uniquely identifies each Cloud, and since they are cloud services, TOS = Cloud; PKA and PKB are public keys for Cloud CA and Cloud CB, respectively. In addition, each Cloud system will have resources such as CPU, Network, Memory and Storage, which it actively advertises to servers.

5.2.3 The Registry

The Registry is a trusted third party that is mostly used to confirm cloud registration and the resource capacity of each cloud. All services and servers must be registered with the Registry including the service requirements to run the service effectively on any Cloud System. It is capable of validating any third-party service, server, or cloud system. It is also an authority in a network used to verify service requests and the identities of all servers in the system. The Registry runs a Resource Allocation Algorithm that does all the mathematical calculations for the Registry to check the total resources and available resources of advertised Cloud Services. The Registry knows the maximum capacity of the Cloud servers with their registered clouds and servers. When the cloud servers send a request for migration to the advertising Cloud, the registry knows the available resources or free resources of the advertising Cloud. The Registry approves the request when it is a registered server and the advertising cloud has enough capacity to provide the services. The Registry knows the requirements to run each service on any Cloud system and therefore can also check when services are acting maliciously, isolated securely, and well-resourced in cloud systems.

Additionally, the Registry knows the Service IDs and Public Keys for each service. A public key that can be used to encrypt and decrypt messages and digital signatures is contained in the digital certificate. The Registry's Public Key is represented here as PKR.

5.2.4 Nonces (N) and Timestamps (T)

Nonces are randomly generated numbers or can occur only once, which have a negligible chance of being guessed by a third party and are used to guarantee the transfer, ensuring that requests cannot be repaid by unauthorized personnel. Timestamps are used for the date and time of an event recorded and explicitly used in migration requests and responses. This allows the system to record how long it takes for migration requests to be granted and how long it takes for migration transfers to complete. In the protocol, timestamps are represented as simple strings.

5.2.5 General Notations

Table 5.1 represents Algorithm 1 naming approach.

Notation	Explanation
CA	Cloud_IDA or Cloud A
CB	Cloud_IDB or Cloud B
SA	Server SA on Cloud CA
SB	New service on Cloud CB
S	Server_ID
ToS	Type of server_ID
PKB/sKB	public and private key pairs of Cloud B
PKS/sKS	public and private key pairs of server SA
PKR /sKR	public and private key pairs of the Registry
NA	Nonce of SA
NB	Nonce of CB
Advertisement	Cloud CB resources
Req.Res	Requested Resources of SA
Res	Resources of CB
Migration_Req /Migration_Resp	Request/Response for resource migration
Migration_Trans	migration transfer
Transfer_Ack	Acknowledgement of migration
Transfer_Complete	Service SB updates its new location to the Registry
TA	Timestamps - SA sends migration request
TB	Timestamps - CB sends migration response
Tcomp	Total time taken for migration complete

Table 5.1: General Notions

5.3 Algorithm1 of first approach

For mobile clients to access Cloud resources and complete their tasks, a solution is required that is constant, secure, and guarantees reliable connectivity. Servers must not end up being hosted on insecure Cloud systems, which can affect or delay the service transfer to mobile clients, and Clouds do not end up hosting insecure servers that can damage Cloud infrastructure. This simulation study addresses these issues by providing a new security protocol (RASP) for secure service migration. In the first approach, the protocol is developed and tested through a simulation study using the Automated Validation of Internet Security Protocols and Applications (AVISPA) tool, which is used to analyse large-scale Internet security protocols and applications. In the RASP protocol, the system moves server SA from Cloud CA to Cloud CB. As we mentioned earlier, the protocol has three entities: server SA on Cloud CA, Cloud CB, and the Registry.

5.4 RASP_Algorithm_V1

The protocol is broken into four stages to clarify the necessary operations involved in secure migration.

5.4.1 Stage 1: Advertisement

Cloud CB advertises its resources and these advertisement are picked up by server SA on Cloud CA. Server SA and Cloud CB do not know anything about each other, and there is no prior communication between SA and CB. In the first step, the server SA receives an advertisement from Cloud CB advertising its resources (ResCB) with its identity CB.

1. CB \rightarrow SA: Advertisement (*Cloud_IDB*, TOS, Res, PKB)

5.4.2 Stage 2: Authentication of SA and CB as well as migration request and response

Both entities, server SA and Cloud CB must be authenticated to each other. Firstly, on receiving an advertisement from CB, SA checks the validity of Cloud CB with the Registry in step 2. So, it sends its identity SA, Cloud CBs identity, a type of server (ToS), the resources of Cloud CB (Res) by using the public key of Registry. In step 3, the Registry will verify the validity of any Cloud resources. The Registry replies to the server SA, saying that Cloud CB is a valid Cloud and encrypts the response with the public key (PKS) of SA, ToS and signs the response with its private key. Therefore, this digital signature ensures that the Registry is the message's originator. In In step 4, SA generates and sends a fresh nonce (NA), its identity SA, ToS, Timestamps (TA), requested resources (Req_Res), and Public key of SA encrypts it using the public key of CB (PKB). Nonces are used to protect this session between the server SA and the Cloud CB to which the server wants to migrate. When CB receives this message in Step 5, it decrypts it using its private key (sKB). Cloud CB verifies the request made by SA by forwarding the requested resources of SA (ResSA), the identity of SA and ToS, to the Registry by using the public key (pKR) of Registry. In Step 6, the Registry replies to forward the requested resources of SA (ResSA), the identity of SA and ToS, to the Registry by using the public key (pKR) of Registry. In Step 6, the Registry replies back to Cloud CB. It validates server SA and encrypts the response with the public key of Cloud CB, and signs it with its private key.

2. SA \rightarrow R: Verify Identity (*CloudID_B*, TOS, PKB, Res, Server_ID, PKS) PKR
3. R \rightarrow SA: Message: YES (*CloudID_B*, TOS, PKB, Valid Res) PKS
4. SA \rightarrow CB: Migration_Req (Server_ID, TOS, TA, Req.Res, PKS, NA) PKB

-
5. CB \rightarrow R: Verify Identity (Server_ID, PKS, TOS, Cloud ID_B, PKB) PKR
 6. R \rightarrow CB: Message: Yes (Server_ID, TOS, Valid Service) PKB

5.4.3 Stage3: Migration transfer

Once the parties authorized each other, Cloud CB sends a message to SA that it can migrate to Cloud CB along with Timestamp TB, acknowledges using unique number NA and generate a new nonce NB in step 7. In step 8, SA migrates the requested resources with server identity, Cloud CB's identity, nonce NB. In step 9, Cloud CB sends an acknowledgement to Server SA with its Identity, CB's Identity, nonce NA, Completion of time calculation (Tcomp) between the transfer. In this protocol, nonces are unique numbers used to protect the sessions between servers and Cloud in migration. This is successfully done using an extended approach based on the Needham Schroeder protocol.

7: CB \rightarrow SA: Migration_Resp (Cloud ID CB, TOS= Cloud), TB, Resources Granted, NA, NB) PKS

8: SA \rightarrow CB: Migration_Tranf (Server ID, CloudID CB, Req resources, NB) PKB

9: CB \rightarrow SA: Transfer Ack (Server ID, CloudID CB, NA, Tcomp) PKS

5.4.3.1 Key Observations

In step 4, the Server generates a new nonce NA as part of the transfer request. This is a unique number generated by the Server side. In step 7, CB generates a new nonce NB, replies to the Server using NA (generated by SA) and NB as part of the transfer response. In step 8, the Server authorizes the service transfer to CB using NB. In step 9, Cloud CB signals that the transfer is completed using NA.

5.4.4 Stage4: Update of New service location to the Registry

The new service SB is now running on Cloud CB. It informs the Registry that it has been successfully migrated and updates the information with SA and CB's Identity, ToS, timestamps TA, TB, Time taken to migrate (Tcomp).

10: SB \rightarrow R: Transf_Comp (Server ID, CloudID B, TOS, TA, TB, Tcomp) PKR

Algorithm 5 Security Protocol for Migration between Server SA on Cloud CA to Cloud CB

- 1: CB \rightarrow SA: Advertisement (*CloudIDB*, TOS, Res, PKB)
 - 2: SA \rightarrow R: Verify Identity (*CloudID_B*, TOS, PKB, Res, Server_ID, PKS) PKR
 - 3: R \rightarrow SA: Message: YES (*CloudID_B*, TOS, PKB, Valid Res) PKS
 - 4: SA \rightarrow CB: Migration_Req (Server_ID, TOS, TA, Req.Res, PKS, NA) PKB
 - 5: CB \rightarrow R: Verify Identity (Server_ID, PKS, TOS, *Cloud ID_B*, PKB) PKR
 - 6: R \rightarrow CB: Message: Yes (Server_ID, TOS, Valid Service) PKB
 - 7: CB \rightarrow SA: Migration_Resp (*Cloud ID_B*, TOS= Cloud), TB, Resources Granted, NA, NB) PKS
 - 8: SA \rightarrow CB: Migration_Tran (Server_ID, *CloudID_B*, Req_resources, NB) PKB
 - 9: CB \rightarrow SA: Transfer_Ack (Server_ID, *CloudID_B*, NA, Tcomp) PKS
 - 10: SB \rightarrow R: Transf_Comp (Server_ID, *CloudID_B*, TOS, TA, TB, Tcomp) PKR
-

- **Step 1** The service on Cloud A receives advertisements from Cloud B advertising Cloud's B resources.
- **Step 2** The Server checks the validity of Cloud B.
- **Step 3** Registry authenticates Cloud B
- **Step 4** The server on Cloud A sends a request to migrate to Cloud B
- **Step 5** Cloud B sends a request to make sure that the server, S, is a valid service:
- **Step 6** The Registry validates Cloud B
- **Step 7** Cloud B signals to The Service on Cloud A that it is OK to migrate the service to Cloud B:
- **Step 8** The Service signals to Cloud B to migrate the service:
- **Step 9** Cloud B signals to the server on Cloud A that the migration is complete: Hence, **New Location: (SA \rightarrow SB)** The service is now started on Cloud B.
- **Step 10** The service on Cloud B signals to the Registry that the migration has been completed:

Figure 5.2 shows the steps for Cloud to Cloud migration of services.

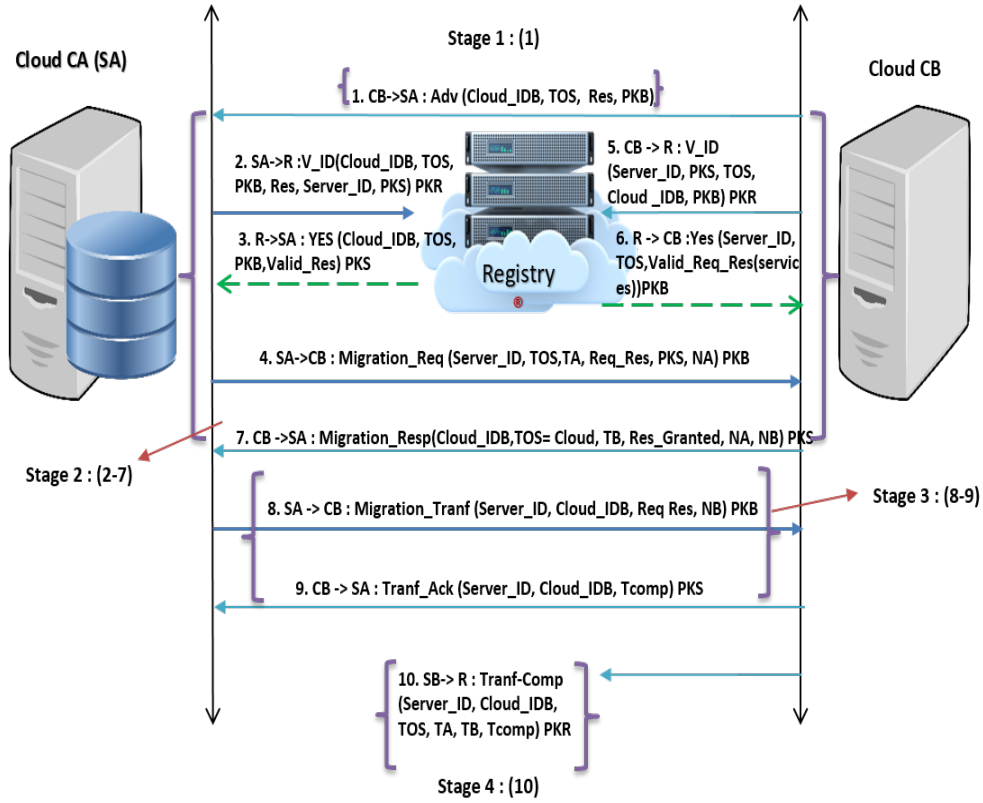


Figure 5.2: Migration from Cloud A to Cloud B

5.5 Evaluation of the First Attempt

5.5.1 HLPSL Specification

The AVISPA framework is role-based and it gives more importance to roles or parties proposed in that protocol than exchanging information. In order to verify the security properties of the security protocol, we had to model three roles: Cloud CA, Cloud CB and the Registry, describing the actions of the protocol. The roles can declare a set of local variables before the initialization of the model checking section, this initialization section is followed by the state transition section. HLPSL defines two composed roles: the session role and the environment role. The HLPSL provides the security properties in GOAL section.

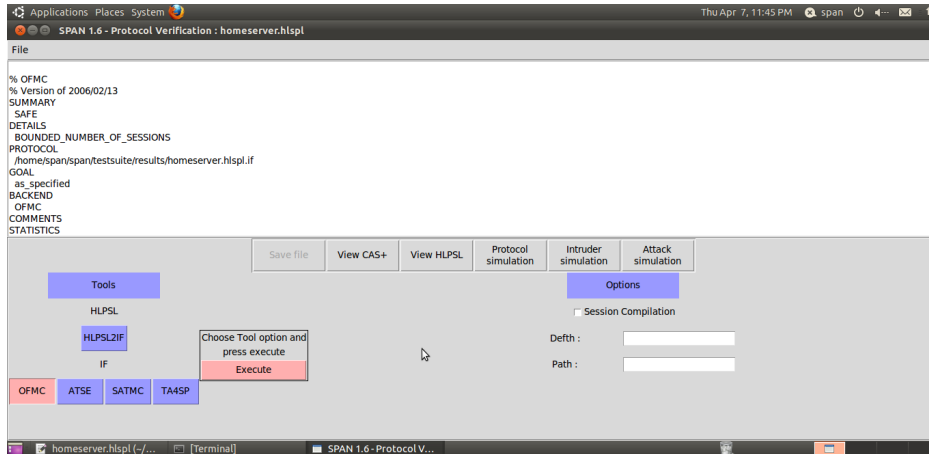


Figure 5.3: OFMC: Cloud A to Cloud B

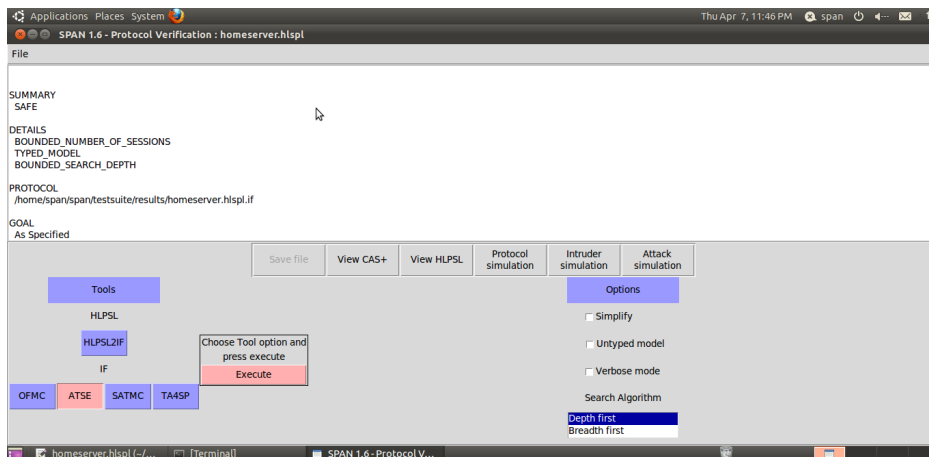


Figure 5.4: ATSE: Cloud A to Cloud B

5.5.2 OFMC and ATSE

After completing the specification as described above, we divided the verification process into two steps. The first step was to validate the specification using OFMC back-end tool of AVISPA, and the second step was to use ATSE back-end. This indicates that our protocol has reached the expected results. The first back end tool is called OFMC. If verifies the combination of two main concepts. In the summary section of the results shown in Figures 5.3 and 5.4 , based on the protocol specification, it indicates that the protocol is SAFE in all communication paths and thus cannot modify the messages, replay messages, or send any false information between parties without authentication. The second back end tool is called ATSE, Constraint-Logic-based Attack Searcher, also indicates that the

protocol is SAFE. This tool handled reducing the redundancies in the protocol and also no protocol falsification for a bounded number of sessions. Now, both of the protocol results SAFE from OFMC and ATSE, so the expected result is accomplished. The protocol satisfies the security properties of secrecy and authentication. In addition, nonces are used for replay protection and to ensure freshness.

In this attempt, this work has demonstrated that a new security protocol for server migration is secured between commercial Cloud environments. The system uses a Registry which validates servers and Cloud infrastructure as well as monitors the migration pattern of services. The protocol is safe under all circumstances; the present protocol critically prevents impersonation attacks either by rogue cloud infrastructure hoping to sneer valid services or by malicious servers wanting to inflict damage on Cloud infrastructure. In our second approach, we will explore how this protocol can be enhanced to prevent intruder and man-in-the-middle attacks by using ProVerif tool.

5.6 Rasp_Algorithm_V2_Second approach

Cloud technologies are increasingly progressing towards interoperability with other clouds. Integrating new security components should be more powerful to secure the migration. As stated previously, the main issues are fraudulent Cloud providers that make service providers host their services on insecure Cloud facilities. It results in data loss, latency and lack of service and misbehaving services that attempt to convince Cloud providers that they are well-behaved systems leading to mismanagement and mishandling of Cloud services. In our second approach, the RASP protocol has been enhanced and broken into four stages, adding a symmetric session key (is used both to encrypt and decrypt information) to secure the transaction. In addition, all the stages of the proposed protocol remain the same but are implemented in a different way and tested by ProVerif. The Figure 5.5 below shows the outline of the second approach.

1. **Stage1: Advertisement:**

Cloud CB actively advertises its resources which is picked up by server SA on Cloud CA.

2. **Stage2: Authentication of SA and CB as well as migration request and response:**

Server SA first requests the Registry to authenticate Cloud CB and the resources it holds. Once it receives the approval from the Registry, it sends a migration request to Cloud CB. Cloud CB then requests the Registry to

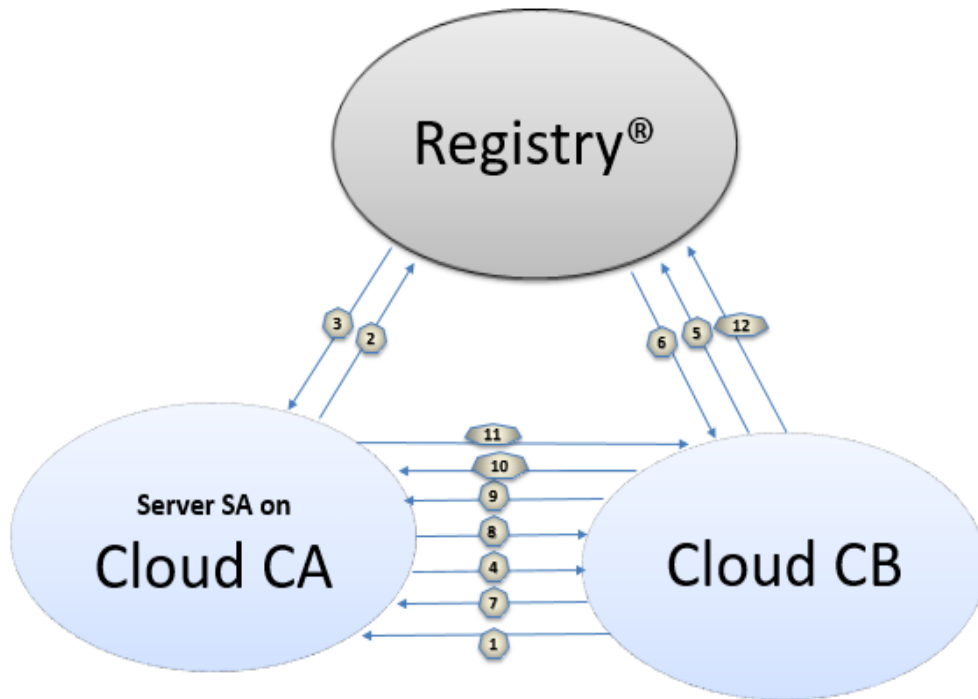


Figure 5.5: Migration from Cloud CA to Cloud CB

authenticate server SA and the resources it requires. Once this is verified, Cloud CB sends a positive migration response to server SA.

3. **Stage3: Migration transfer:**

Server SA sends a message to Cloud CB to begin the migration transfer. Cloud CB begins the transfer and signals server SA when the transfer is completed.

4. **Stage4: Update of New service location to the Registry:**

The new service SB is now running on Cloud CB and informs the Registry that it has been successfully migrated.

5.6.1 Using Symmetric key

In our second attempt, all the stages of the proposed protocol remain the same but implemented in a different way and tested by ProVerif.

5.6.1.1 General Notations

Table 5.2 represents Algorithm 2 - general notations.

Notation	Explanation
CA	Cloud A
CB	Cloud B
SA	Server SA on Cloud A
SB	New service on Cloud B
Ksc	Symmetric session key
pkC/skC	public and private key pairs of Cloud B
pkS/skS	public and private key pairs of server SA
pkR/skR	public and private key pairs of the registry
Ns	Nonce of SA
Nc	Nonce of CB
AdvC	Advertisements
ResSA	Requested Resources of SA
ResCB	Resources of CB
M_Reqc	Request for migration
M_Trfs	migration transfer
M_Ackc	Acknowledgement of migration
sign	Signature/signed by the Registry
aenc	Asymmetric Encryption
enc	Symmetric Encryption

Table 5.2: General Notations

5.6.2 RASP for Migration between server on Cloud CA to Cloud CB

As we stated earlier, the Algorithm 1 and Algorithm 2 remain the same but in Stage 3, we create a secure session key (Ksc) which is used for sessions between server SA and Cloud CB. In addition, the same key will be used for encryption and decryption. Server SA generates the session key (Ksc), to start the migration request (M_Reqc). By using the session key (Ksc), Cloud CB begins the transfer (M_Trfs) and signals server SA when the transfer is completed (M_Ackc).

In this second attempt, Stage 1 corresponds to step 1 of the protocol; Stage 2 corresponds to steps 2-7; Stage 3 corresponds to 8-11, and finally, Stage 4 corresponds to step 12. The RASP protocol is followed in exactly the same way

an outlined below and Figure 5.6 shows the steps for Cloud-to-Cloud migration of services.

- **Stage 1**
 1. $CB \rightarrow SA : Advc (CB, ResCB)$
- **Stage 2**
 2. $SA \rightarrow R : (SA, CB, ResCB)pkR$
 3. $R \rightarrow SA : sign((CB, pkC, ResCB), pkS)$
 4. $SA \rightarrow CB : aenc ((Ns, SA, ResSA), pkC)$
 5. $CB \rightarrow R : (CB, SA, ResSA)pkR$
 6. $R \rightarrow CB : sign((SA, pkS, ResSA), pkC)$
 7. $CB \rightarrow SA : aenc((Ns, Nc, CB), pkS)$
- **Stage 3**
 8. $SA \rightarrow CB : aenc((Nc, Ksc), pkC)$
 9. $CB \rightarrow SA : enc ((M_Reqc, SA), Ksc)$
 10. $SA \rightarrow CB : enc ((M_Trfs, ResSA), Ksc)$
 11. $CB \rightarrow SA : enc ((M_Ackc), Ksc)$
- **Stage 4**
 12. $SB \rightarrow R : aenc((SB, CB), pkR)$

5.6.3 The RASP Protocol in detail

- **Step 1** Server SA receives advertisements from Cloud B advertising Cloud B's resources with its identity CB.
- **Step 2** Server SA checks the validity of Cloud CB with the Registry.
- **Step 3** The Registry authenticates server SA.
- **Step 4** Server SA sends a migration request to Cloud CB.
- **Step 5** Cloud CB sends a request to the Registry to make sure that server SA's request for resources is valid.
- **Step 6** The Registry replies back to Cloud CB.
- **Step 7** Cloud CB sends the migration response back to server SA.
- **Step 8** Server SA generates the session key (Ksc) to start the migration request.
- **Step 9** Cloud CB sends the migration initialisation request.

- **Step 10** Server SA does the migration transfer.
- **Step 11** Cloud CB sends an acknowledgement to server SA.
- **Step 12** Service on Cloud B (SB) updates the Registry on its new location.

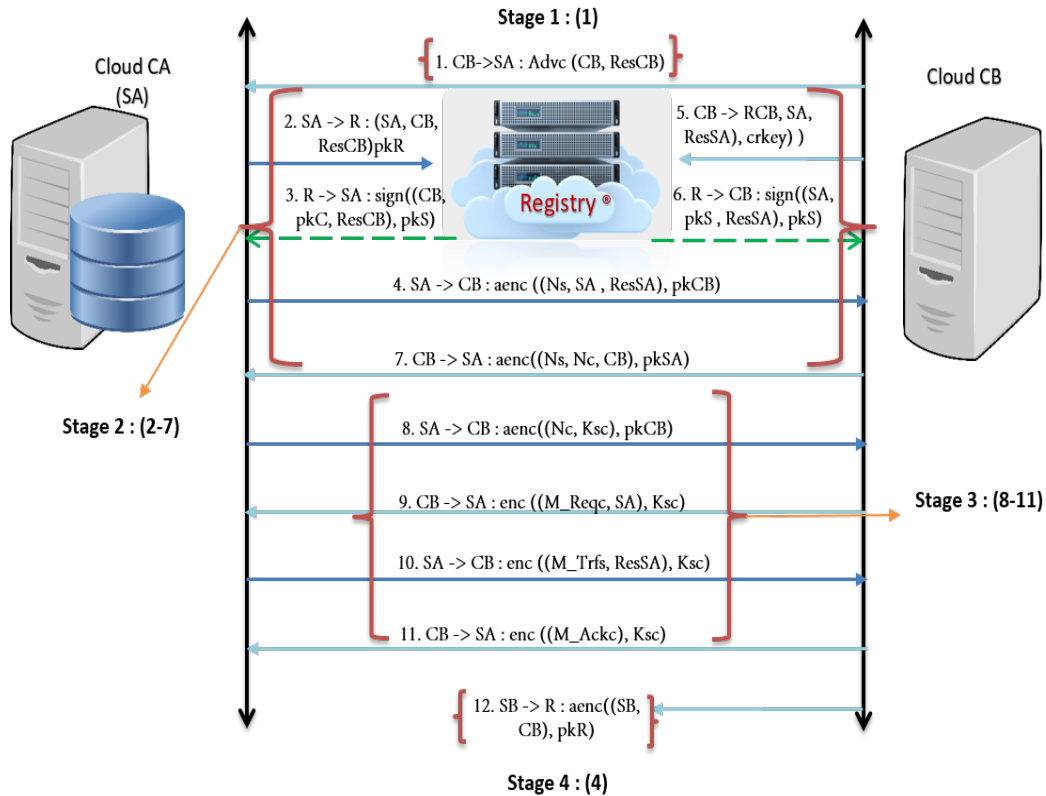


Figure 5.6: Second Attempt Migration between SA to CB

5.6.4 Stages in detail

5.6.4.1 Stage 1

Cloud CB advertises (Advc) its resources which are picked up by server SA on Cloud A. Server SA and Cloud CB do not know anything about each other and there is no prior communication between SA and CB. In the first step, the server SA receives an advertisement from Cloud B advertising its resources (ResCB) with its identity CB.

1. CB → SA : Advc (CB, ResCB)

5.6.4.2 Stage 2

Both entities, server SA and Cloud B, must be authenticated to each other. Firstly, on receiving an advertisement from CB, SA checks the validity of Cloud B with the Registry. So it sends its identity SA, Cloud B's identity, CB, and the resources at Cloud B (ResCB) by using the public key of Registry. The Registry will be able to verify the validity of any Cloud resources. The Registry replies back to server SA, saying that Cloud CB is a valid Cloud and encrypts the response with the public key(pkS) of SA and signs the response with its private key. Therefore, this digital signature ensures that the Registry is the originator of the message. SA generates and sends a fresh nonce (Ns), its identity SA, requested resources (ResSA) and encrypts it using the public key of Cloud B, (pkC). Nonces are used to protect this session between the server SA and the Cloud B to which the server wants to migrate. When CB receives this message, it decrypts the message using its private key (skC). Cloud CB verifies the request made by SA by forwarding the requested resources of SA (ResSA), the identity of SA and its identity, CB, to the Registry by using the public key (pkR) of Registry. The Registry replies back to Cloud CB, it validates server SA and encrypts the response with Cloud CB's public key and uses its private key to sign it. Finally, Cloud B sends a message to server SA that it can migrate to Cloud B, it acknowledges Ns and generates a new nonce, Nc.

2. SA \rightarrow R : (SA, CB, ResCB)pkR
3. R \rightarrow SA : sign((CB, pkC, ResCB), pkS)
4. SA \rightarrow CB : (Ns, SA, ResSA), pkC
5. CB \rightarrow R : (CB, SA, ResSA)pkR
6. R \rightarrow CB : sign((SA, pkS, ResSA), pkC)
7. CB \rightarrow SA : aenc((Ns, Nc, CB), pkS)

5.6.4.3 Stage 3

In Stage 3, server SA generates a fresh symmetric session key (Ksc), pairs it with the nonce generated by CB (Nc), encrypts the message by using Cloud B public key (pkC) and sends to CB. Cloud B initiates the migration request (M_Reqc), pairs it with server SA, and encrypts it using the generated symmetric session key (Ksc). Once the migration request from CB to SA is received, SA is able to decrypt it using the shared key (Ksc) and starts the migration transfer (M_Trfs) with the requested resource (ResSA) by using the symmetrical session key (Ksc). Once the migration transfer is completed, CB sends an acknowledgement (M_Ackc), to server SA which is encrypted using Ksc.

8. SA \rightarrow CB : aenc((Nc, Ksc), pkC)

-
9. $CB \rightarrow SA : \text{enc}((M_Reqc, SA), Ksc)$
 10. $SA \rightarrow CB : \text{enc}((M_Trfs, ResSA), Ksc)$
 11. $CB \rightarrow SA : \text{enc}((M_Ackc), Ksc)$

5.6.4.4 Stage 4

In Stage 4, the service (SB) updates the Registry on its new location and hence $SA \rightarrow SB$.

12. $SB \rightarrow R : \text{aenc}((SB, CB), pkR)$

5.7 Modelling the Protocol by using ProVerif

Generally, protocols themselves are described by a set of roles, each role with an associated script or sequence of events. In ProVerif, the input language will follow these three steps. The above protocol has been designed using a ProVerif tool. Parties are modelled as processes, and the attackers need not be specified. When modelling a protocol in ProVerif, there are three parts. In Part 1, the declarations formalise the behaviour of security mechanisms or crypto primitives in that security protocol; in Part 2, the process macros section allows the list of sub-processes to be defined, to have sessions between server and cloud; and finally, in Part 3, the protocol has the main process with the use of macros. This is where the protocol analysis has begun.

5.7.1 Part1: Declarations

The process macros contain a finite set of variables, constants, types, free names and constructor and destructor functions used to develop security mechanisms. ProVerif declares free name “ch” of type communication channel which will be used for public communications (e.g., Internet), free name S, C as two honest hostnames Server SA and Cloud CB (e.g. Participants IDs, etc.). Requested resources (Ress) and Cloud advertising resources (Resc) are declared as free name Ress, Resc type of bit string. The keyword `private` excludes the names from the attacker’s knowledge (e.g. Secret keys). The internal representation of Keys and nonces declares as predefined type keys, nonce. For example, public and private keys declare as type pkey, type skey and the constant is an instance for names. For example, it uses for advertisement (Adv) as `const Adv`, Migration response (MReqc), Migration Acknowledgement (MAckc) and Migration transfer (MTrfs). The table keyword `store` records, keys (table keys (host, pkey)) 5.7.

```

(* ***** DECLARATIONS ***** *)

free ch: channel.      (* Public channel*)
type host.            (* Host names C, S, SB *)
type nonce.          (* Random Gnerated numbers Nc, Ns*)
type id.

(* Two honest host names Server(SA),Cloud(CB)*)
free S, C: host.

(* Requesting resources of Server SA *)
free Ress: bitstring [private].

(* Resources of Cloud CB *)
free Resc: bitstring.

const Adv: id.
const MReqc: id.
const MTrfs: id.
const MAckc: id.

(* Key types *)
type pkey.
type skey.
type spkey.          (* Signed public key *)
type sskey.         (* Signed private key *)
type key.           (* Session key *)

(* Table *)
table keys(host, pkey).

```

Figure 5.7: ProVerif - Declarations

5.7.2 Part1a: Modelling Constructor and Destructor for Crypto primitives

The secure service migration is between Server SA (S) and Cloud CB(C) in the resource allocation security protocol. We define the following security mechanisms to our protocol. Functions are divided into two parts such as constructor and destructor. The constructor helps build the terms. Thus, we have the constructor form of constructor-name (m1,m2...mn). On the other hand, destructor functions are used to manipulate the messages (terms) in the process. Thus, we have the destructor form of destructor-name (m1, m2...mn). Combined constructor and destructor functions are used to capture the relationship between the security mechanisms. Thus, we have used the security mechanisms to capture the secrecy(confidentiality of the message), authentication (correspondence assertions between parties), and key exchange (secure transaction to migrate the services).

```

(* ***** CRYPTO MODELS ***** *)

(* Asymmetric key encryption *)
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall x: bitstring, y: skey; adec(aenc(x, pk(y)), y) = x.

(* Signatures *)
fun spk(sskey): spkey.
fun sign(bitstring, sskey): bitstring.
reduc forall m: bitstring, k: sskey; getmess(sign(m, k)) = m.
reduc forall m: bitstring, k: sskey; checksign(sign(m, k), spk(k)) = m.

(* Symmetric encryption *)
fun enc(bitstring, nonce): bitstring.
reduc forall x: bitstring, y: nonce; dec(enc(x, y), y) = x.

(* Session key encryption *)
fun symenc(bitstring, key): bitstring.
reduc forall x: bitstring, y: key; symdec(symenc(x, y), y) = x.

```

Figure 5.8: ProVerif - Constructor and Destructor for Crypto primitives

5.7.2.1 Symmetric Encryption

A single key is used in symmetric encryption to encode and decode the message. The term name "fun" refers to the constructor function, and the term "enc" refers to the name of the constructor, which passes through two arguments or binary operators and type of "bitstring". The term "reduc" refers to the destructor function; it uses the same key associated with the initiator messages to decrypt the cypher text.

- fun enc(bitstring, nonce): bitstring.
- reduc forall x: bitstring, y: nonce; dec(enc(x,y),y) = x.

5.7.2.2 Asymmetric key Encryption

This encryption techniques use public(pkey) and private keys(skey) to encrypt and decrypt the messages.

- fun pk(skey): pkey.
- fun aenc(bitstring, pkey): bitstring.
- reduc forall x: bitstring, y: skey; adec(aenc(x, pk(y)),y) = x.

5.7.2.3 Session key encryption

This session key encryption is used to randomly generate an encryption key between server SA and cloud CB to do the migration transfer, and this key exists until the session or transmission ends. It uses the same key, sometimes referred to as Symmetric session key or symmetric key.

- fun shaenc(bitstring,key): bitstring.
- reduc forall x: bitstring, y: key; symdec(symenc(x,y),y) = x.

5.7.2.4 Signatures

Signature primitives help for verifying the authenticity of the messages.

- fun spk(sskey): spkey.
- fun sign(bitstring, sskey): bitstring.
- reduc forall m: bitstring, k: sskey; getmess(sign(m,k)) = m.
- reduc forall m: bitstring, k: sskey; checksign(sign(m,k), spk(k)) = m.

5.7.3 Part2: Process macros

The processes are defined using sub-processes, containing conditionals, let-definitions and pattern matching tuples. Figure 5.9 shows a Server SA process, Cloud CB processes 5.10, and 5.11 shows trusted party processes. It is defined by the macro process, starting with "let process" definitions. The term "in ()" and "out ()" refers to receiving and sending messages. The keyword "new" generates a new value in protocol sessions. Events can be inserted into the process to verify correspondence assertions, for example, event beginCparam (hostS) and event endSparam (S). By using cryptographic primitives, the requested resources are successfully migrated between SA and CB.

```

(* ***** SERVER SA PROCESS-hostY ***** *)
let processS(pkR: spkey, skS: skey, pkC: pkey, srkey: key) =
  in(ch, (Adv: id, hostX: host, Resc: bitstring));
  (* SA receives advertisements from Cloud CB advertising its resource:
  Adv(CB, ResCB) *)

  event beginCparam(hostX);
  out(ch, (S, hostX, Resc));
  (* SA checks the validity of CB: (SA, CB, ResCB)*)

  in(ch, m1: bitstring);
  (* The Registry authenticates CB :m1 = signed(CB, public key of CB, Resources) *)

  let (pkX: pkey, =hostX, Resc: bitstring) = checksign(m1, pkR) in
  (* The destructor checksign checks the signature and returns the message *)

  new Ns: nonce;
  (* Nonce Ns Random Number Generation *)

  out(ch, aenc((Ns, S, Resc), pkX));
  (* Server SA sends a request to migrate to Cloud CB: aenc((Ns, SA , ResSA), pkCB)*)

  in(ch, m3: bitstring);
  (* SA receives migration response:(Ns, Nc, CB), pkS) *)
  let (=Ns, NX: nonce , =hostX) = adec(m3, skS) in
  (* Identity verified *)
  new Ksc : key;
  (* Generates session key to do the migration transfer *)
  out(ch, aenc((nonce_to_bitstring(NX), Ksc), pkX));
  in(ch, m6: bitstring);
  let (MReqc: id, =S) = symdec(m6, Ksc) in
  out(ch, symenc((MTrfs, Resc), Ksc));
  in(ch, m8: bitstring);
  let (MAckc: id, =S) = symdec(m8, Ksc) in
  if hostX = C then
  event endSparam(S);
  out(ch, enc(SNs, Ns));
  out(ch, enc(SNc, NX));

```

Figure 5.9: Server SB processes

5.7.4 Part3: main processes

The Figure 5.12 is a main macro process used for calling let functions and running the number of times based on the protocol sessions. The main process also has conditionals, let definitions, and pattern matching tuples. Additionally, it has the parallel composition of Server SA and Cloud CB, used to represent hosts of a protocol running in parallel. The replication“!” is an infinite composition for the unbounded sessions between them. The processes are executing multiple times by using the following syntax: (!processS (pkR, skS, pkC, srkey)) Parallel (!processC(pkR, skC, pkS, crkey)) Parallel (!processR (skR, srkey, crkey))

```

(* Cloud - host X *)
let processC(pkR: spkey, skC: skey, pkS: pkey, crkey: key) =
  out(ch, (Adv, C, Resc));
  in(ch, m2: bitstring); (* Cloud Receives Ns *)
  let (NY: nonce, hostY: host, Ress: bitstring) = adec(m2, skC) in
  event beginSparam(hostY);
  out(ch, symenc((C, hostY, Ress), crkey)); (* Cloud to registry requests for S Public key*)
  in(ch, m4: bitstring); (* Registry to cloud R->C : S,pks *)
  let (pkY: pkey, = hostY, Ress: bitstring) = checksign(m4, pkR) in
  new Nc: nonce; (* Nonce Nc creation*)
  out(ch, aenc((NY, Nc, C), pkY)); (* Cloud sends ns,nc,C to server *)
  in(ch, m5: bitstring); (* m3 variable has nonce nc and session key*)
  let (= Nc, Ksc:key) = adec(m5, skC) in
  out(ch, symenc((MReqc, hostY), Ksc));
  in(ch, m7: bitstring);
  let (MTrfs: id, Ress: bitstring) = symdec(m7, Ksc) in
  out(ch, symenc((MAckc, hostY), Ksc));
  out(ch, (Ress, C));
  if hostY = S then
  event endCparam(C);
  out(ch, enc(CNs, NY));
  out(ch, enc(CNc, Nc));
  out(ch, symenc(CNk, Ksc));
  out(ch, symenc(CNk, crkey)).

```

Figure 5.10: Cloud CB processes

```

(* Trusted key server *)
let processR(skR: sskey, srkey: key, crkey: key) =
  in(ch, (s: host, c: host, Resc: bitstring, Ress: bitstring));
  get keys(= c, pc) in
  if (Resc = Ress) then
  out(ch, sign((pc, c, Resc), skR)). (*rc- public key of C and c- cloud host = pkX: pkey, =hostX *)

```

Figure 5.11: Registry server

```

(* ***** MAIN Process ***** *)

new skS: skey;
new srkey: key;
new crkey: key;
let pkS = pk(skS) in out(ch, pkS); insert keys(S, pkS);
new skC: skey;
let pkC = pk(skC) in out(ch, pkC); insert keys(C, pkC);
new skR: skey; let pkR = spk(skR) in out(ch, pkR);

```

Figure 5.12: Main Process Macros

5.7.4.1 Evaluation of the Second Attempt

In our second attempt, by using the symmetric session key (Ksc), the requested service is transferred to the new location CB. As we mentioned in the first attempt, nonces protect the session between server SA and Cloud CB. However, in the second attempt, the symmetrical session key is used to do the actual transfer. Hence, this is a more secure mechanism than using nonces for the actual data transfer.

5.7.5 ProVerif Results

The results show that RASP can achieve the secrecy, authentication and key exchange of the service migration mechanism. We verified Stage 2 and Stage 3 using the ProVerif tool and presented the programme's output, which is given in Figure 5.13 below. The security properties were specified in the input language to check whether the property is derivable by the attacker or not [Karthick et al. \(2018\)](#).

Results in Terminal:ProVerif/opam/system/bin/proverif1.98pl1 docs/Nee/CB toSA_SecuiryProtocol.pvgrep "RES".

5.7.5.1 Nonces are secured and not derived by the attacker

- RESULT not attacker_bitstring (SNs []) is true.
- RESULT not attacker_bitstring (SNc []) is true.
- RESULT not attacker_bitstring (CNs []) is true.
- RESULT not attacker_bitstring (CNc []) is true.

5.7.5.2 The session key is not derived by the attacker

- RESULT not attacker_key (Ksc [m3 = v_1975, m1 = v_1976, hostX = v_1977! 1 = v_1978]) is true.

```

gayathri@pl-00108268: ~/opam/system/bin
Starting query not attacker_bitstring(CNc[])
RESULT not attacker_bitstring(CNc[]) is true.
-> Query not attacker_bitstring(SNk[]); not attacker_bitstring(CNk[])
Completing...
Starting query not attacker_bitstring(SNk[])
RESULT not attacker_bitstring(SNk[]) is true.
Starting query not attacker_bitstring(CNk[])
RESULT not attacker_bitstring(CNk[]) is true.
-> Query not attacker_key(Ksc[m3 = v_1975,m1 = v_1976,hostX = v_1977,i1 = v_1978])
Completing...
Starting query not attacker_key(Ksc[m3 = v_1975,m1 = v_1976,hostX = v_1977,i1 = v_1978])
RESULT not attacker_key(Ksc[m3 = v_1975,m1 = v_1976,hostX = v_1977,i1 = v_1978]) is true.
-- Query not attacker_skey(skC[])
Completing...
Starting query not attacker_skey(skC[])
RESULT not attacker_skey(skC[]) is true.
-- Query not attacker_skey(skS[])
Completing...
Starting query not attacker_skey(skS[])
RESULT not attacker_skey(skS[]) is true.
-> Query inj-event(endSparam(x_4663)) ==> inj-event(beginSparam(x_4663))
Completing...
Starting query inj-event(endSparam(x_4663)) ==> inj-event(beginSparam(x_4663))
goal reachable: begin(beginSparam(S[]), m4 = sign(pk(skS[]),S[]),skR[]), m2 = aenc((Ns[m1 = sign((pk(skC[]),c[]),skR[]),hostX = c[],i1 = endsid_5751],S[]),pk(skC[])), @tsid_604 = @tsid_5752, @occ32 = @occ_cst) -> end(endsid_5751,endSparam(S[]))
RESULT inj-event(endSparam(x_4663)) ==> inj-event(beginSparam(x_4663)) is true.
-> Query inj-event(endCparam(x_5762)) ==> inj-event(beginCparam(x_5762))
Completing...
Starting query inj-event(endCparam(x_5762)) ==> inj-event(beginCparam(x_5762))
RESULT inj-event(endCparam(x_5762)) ==> inj-event(beginCparam(x_5762)) is true.
gayathri@pl-00108268:~/opam/system/bin$ ./prover1f -f./opam/system/bin/prover1f1.98pl1/docs/INCTT/CBtoSA_SecurityProtocol.pv | grep "RES"
RESULT not attacker_bitstring(SNs[]) is true.
RESULT not attacker_bitstring(CNs[]) is true.
RESULT not attacker_bitstring(CNc[]) is true.
RESULT not attacker_bitstring(SNk[]) is true.
RESULT not attacker_bitstring(CNk[]) is true.
RESULT not attacker_key(Ksc[m3 = v_1975,m1 = v_1976,hostX = v_1977,i1 = v_1978]) is true.
RESULT not attacker_skey(skC[]) is true.
RESULT not attacker_skey(skS[]) is true.
RESULT inj-event(endSparam(x_4663)) ==> inj-event(beginSparam(x_4663)) is true.
RESULT inj-event(endCparam(x_5762)) ==> inj-event(beginCparam(x_5762)) is true.
gayathri@pl-00108268:~/opam/system/bin$

```

Figure 5.13: Second Attempt results using Proverif

- RESULT not attacker_bitstring (SNk []) is true.
- RESULT not attacker_bitstring (CNk []) is true.

5.7.5.3 Private keys of SA & CB are not derived by the attacker

- RESULT not attacker_skey (skC []) is true.
- RESULT not attacker_skey (skS []) is true.

5.7.5.4 Authentication SA to CB and CB to SA is true

- RESULT inj-event (endSparam (x_4663)) → inj-event (beginSparam (x_4663)) is true.
- RESULT inj-event (endCparam (x_5762)) → inj-event (beginCparam (x_5762)) is true.

In our second attempt, by using symmetric session key (Ksc), the requested service is transferred to the new location CB. As we mentioned in the first attempt, nonces are used to protect the session between server SA and Cloud CB. However, in the second attempt, the symmetrical session key is used to do the actual transfer. Hence, this is a more secure mechanism than using nonces for the actual data transfer. Table number RASP protocol query attacker () results

5.7.6 Query attacker ()

Table 5.3 represents query results

Security properties	Server SA event	Cloud CB event
Session key	TRUE	TRUE
Private keys	TRUE	TRUE
Nonces	TRUE	TRUE
Event begin	TRUE	TRUE
Event begin	TRUE	TRUE
Requested resources	TRUE	-
Advc Resources	-	False

Table 5.3: Query attacker of results

5.8 Chapter Summary

In our second attempt, by using symmetric session key (Ksc), the requested service is transferred to the new location CB. As we mentioned in the first attempt, nonces are used to protect the session between server SA and Cloud CB. However, in the second attempt, the symmetrical session key is used to do the actual transfer. Hence, this is a more secure mechanism than using nonces for the actual data transfer. This research explored the development of a new resource allocation and secure service migration framework that encompasses the computing resources to migrate in cloud environments. RASP, in turn, will use standard migration mechanisms such as Docker, KVM, LXD, and Unikernels to do the actual migration. Hence, secure resource allocation algorithms must be deliberated as a main galvanizing structure to provide seamless connectivity and maintain QoS in highly mobile Environments such as VANETs.

Chapter 6

New mechanisms to provide more security in Service Environments

6.1 Brief Introduction

This chapter describes new mechanisms such as the Capabilities based system and secondly, a New Service Management Framework (SMF) for mobile clouds or Vehicular Networks.

6.2 Capabilities

A Capability refers to an unforgeable token or a key of authority that permits authorized users to access specific objects in a system or references an object with associated rights. A capability is implemented as a data structure containing two pieces of data: a unique object identification and access rights. They can provide Role-Based Access Control (RBAC) access for users. Some capabilities are therefore not directly assigned to users; instead, they are assigned to roles, and roles are assigned to users. These are called role-based capabilities. The IPv6 address space, based on a modified Location/ID split and is based on the work of [Mapp et al. \(2011\)](#), enables the creation of capability ID-based system for people, apps, and cloud infrastructure. As we know, Capabilities are used to identify objects and their properties and thus, they must be carefully managed and protected against being generated or altered inappropriately.

6.2.1 Capability structure

The created capability-based system is built on IPv6's flexible approach, which allows communication between objects using both unique ID and location as

a method. In particular, the object field for capabilities uses a modified IPv6 Location/ID split and is based on the work of [Mapp et al. \(2011\)](#). In this system the ID is composed of an EUI which is 64-bit and an 8-bit netadmin field that is used to manage the object. This enhances both performance and security. Capabilities can be used to identify any object and its properties because the capabilities cannot be forced without being detected. Therefore, they can be directly used to refer to associated objects. As a result, capabilities must be monitored and secured from being produced or updated without authorization. The format of a capability-based system is shown in [Figure 6.1](#).

TYPE	SYS FIELD	PROPERTY FIELD	OBJECT ID	RANDOM BIT FIELD	HARSH FIELD
------	--------------	-------------------	--------------	---------------------	----------------

Figure 6.1: Capability Structure

6.2.1.1 Type Field (8 bits)

This field is used to specify the object capability that is currently being used. Users, digital assets, and facilities are examples of types. The type field also has a TIMED type, which is used to indicate a timed-based capability that is only valid for a set amount of time beyond which the system refuses to assign any rights to the holder. A special object type called as a Capability List (CL) has also been designed to aid with system administration. The CL is used to the group together a set of capabilities.

6.2.1.2 SYS Field (4 bits)

This SYS field is used to provide what is necessary to accomplish a task in managing capabilities. The capability related fields are given by four bits. SYS FIELD is shown in [Figure 6.2](#).

P	S	M	C
---	---	---	---

Figure 6.2: Capability Structure

1. The Private(P) bit is used to limit the number of persons who have access to the capability.
2. The System bit(S) specifies whether the object in question was made by the system, an application, or a user. This means that users or apps cannot

change or remove the capability that the system has created. But, the system can remove or modify the capabilities of users and applications when necessary.

3. The Master(M) bit refers that the capability was created by a Certificate Authority (CA) or Registry.
4. The Change(C) bit is used to indicate if this capability can be changed. This means that if this bit is set, the proxy capabilities can be derived from the master capability.

6.2.1.3 Property Field (12 bits)

This Property Field is used to specify the object's properties. The field relates to the attributes or functions of the item that the capability represents in this field.

6.2.1.4 Object ID (72 bits)

This field is used to uniquely identify an object in the system and used a 64-bit EIU field and an 8-bit netadmin field.

6.2.1.5 Random Bit Field (16 bits)

The random field guarantees unforgeability. This field helps identify the object uniquely. The random bit field is generated after the type field, sys field, property field, and object ID field are created. When proxy certificates are created, a new random field is generated.

6.2.1.6 Hash Field (16 bits)

The hash field is used to allow the detection of the tampering of capabilities. When a capability is created, the type field, sys field, property field, and object ID field are first generated, followed by the random bit field. Finally, these fields are used to generate a SHA-1 hash which is placed in the hash field of the capability.

6.2.2 Rules for Capabilities

The capability based system is very flexible and thus, additional rules are needed to ensure proper usage when generating.

1. When an object is created for the first time, a capability is established for it. This is known as a master capability, and the object's owner is marked as the master capacity's owner.

-
2. It is possible to create proxy capabilities from the master capability if the change bit in the master capability is set.
 3. Proxy capabilities must have a new random bit field and a new hash field.
 4. Proxy capabilities cannot be created from other proxy capabilities. This rule is necessary to prevent the creation of capabilities by unauthorised users from gaining access to capabilities.
 5. System capabilities cannot be generated or changed by users. This rule is needed to protect key entities such as operating systems as well as access to system services.

The system can be used to deliver the necessary security requirements for many environments, including healthcare systems, Cloud computing systems, and the Internet of Things (IoT). Capabilities can help secure real systems such as health-care systems [Vithanwattana et al. \(2021a\)](#).

6.3 New Service Management Framework

Addressing the security challenges in real world requires a new approach to service delivery that can consider security, deployment, replication, or service migration issues on a regional, national, or global scale. The development of highly mobile environments such as VANETs, Mobile Edge Computing, and the Internet of Things requires several interfaces to connect seamlessly and provide mobile users with excellent performance. Mobile Edge Computing brings the computing capabilities closer to the edge of the network or near the end-user or server. This service minimizes the need for data processing on remote servers, reduces latency, and increases bandwidth. However, it is crucial to enable service migration to support these networks, which will migrate as mobile users move around. The new environment is shown in [Figure 6.3](#).

Though we need a robust service architecture to cover such features, a new service management framework for mobile clouds has been developed ([Jose, 2020](#)). This service architecture allows services to be managed, copied, or migrated to support mobile users as they move around.

The requirement to enable high Quality of Service (QoS), imposes the development of a new Service Management Framework that allows services to migrate to different locations while maintaining a good QoS for mobile users. This chapter identifies main contributions of the research which will focus on developing the Service Management Prototype in the proposed security framework for systems.

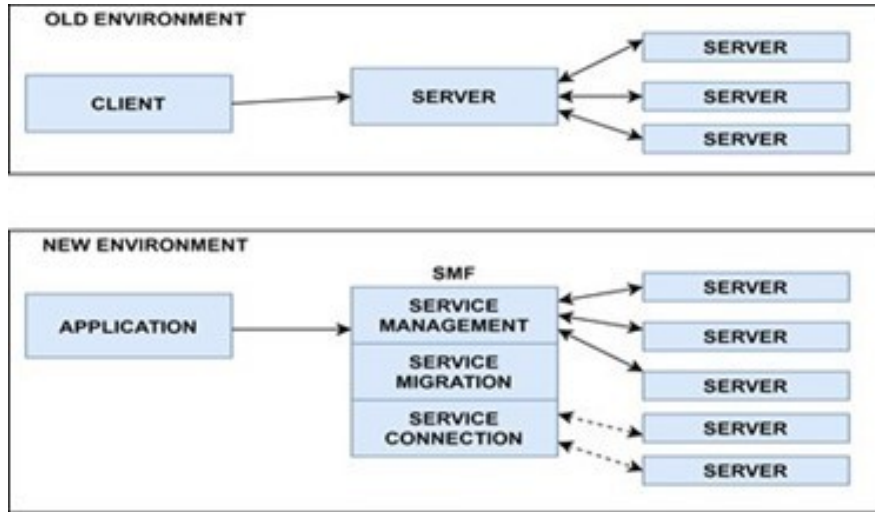


Figure 6.3: Effect of introducing SMF in the client-server environment

The new proposed implementation framework is based on the reference framework developed by Sardis and has four layers. The overall structure is shown in Figure 6.4.

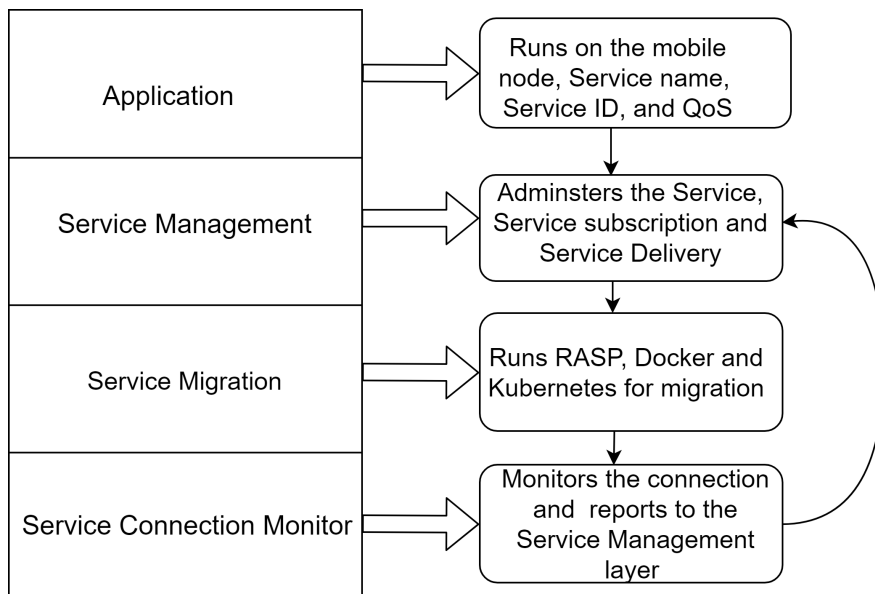


Figure 6.4: Service Migration Prototype

- **Application Layer (AL):** This is the first layer of SMF and runs on the mobile node and invokes the service through the Service Management

Layer (SManL), giving the service name, Service ID and required QoS. When service comes to the SMF for the first time, it should have a unique ID to be identified. The service name provides which type of service is running, such as CPU, Memory or Storage. Finally, the Application Layer must define QoS parameters for the required service such as delay time or latency, bandwidth, reliability and security.

- **Service Management Layer (SManL):** This layer is the management layer that administers the mobile service and is also responsible for the service subscription and service delivery for the below layers. The service subscription refers to Service-Level Agreement (SLA) and billing arrangements. Service delivery describes how services should migrate from one location to another. Once this layer decides that a service should be migrated, it passes this information to the service migration layer (SML).
- **Service Migration Layer (SML):**] This service migration layer handles the migration requested by SManL and uses a Resource Allocation Security Protocol (RASP) for secure migration to ensure that the transfer is securely done. SML updates SManL when the migration is completed. In turn, RASP Karthick et al. (2018) will use standard migration mechanisms such as Docker, KVM, LXD and Unikernels to do the actual migration.
- **Service Connection Layer (SCL):** This layer monitors how clients connect to the services between the mobile node and the server. It reports to the SManL when the mobile node is no longer available due to handover to another network.

6.4 SSP Protocol - An overview

Capabilities can be used flexibly to provide AAA in many environments. Additionally, by combining capabilities, SMF and the techniques used in RASP, it is possible to design a Secure Service Protocol (SSP) that can be used to protect any service. In traditional computer systems, we have client-server communications. The SSP protocol has roles such as client, server, and Service Management Framework (SMF). It has been developed to ensure that the proposed service protocol is safe and can be applied to ANY service. The protocol is broken into five stages to clarify the necessary operations in secure service migration. The interaction between an application, the Service Management Framework (SMF) and services supported by the SMF are shown in Table 6.1 and Figure 6.5. The SSP protocol is defined in the same way that the RASP protocol as outlined in

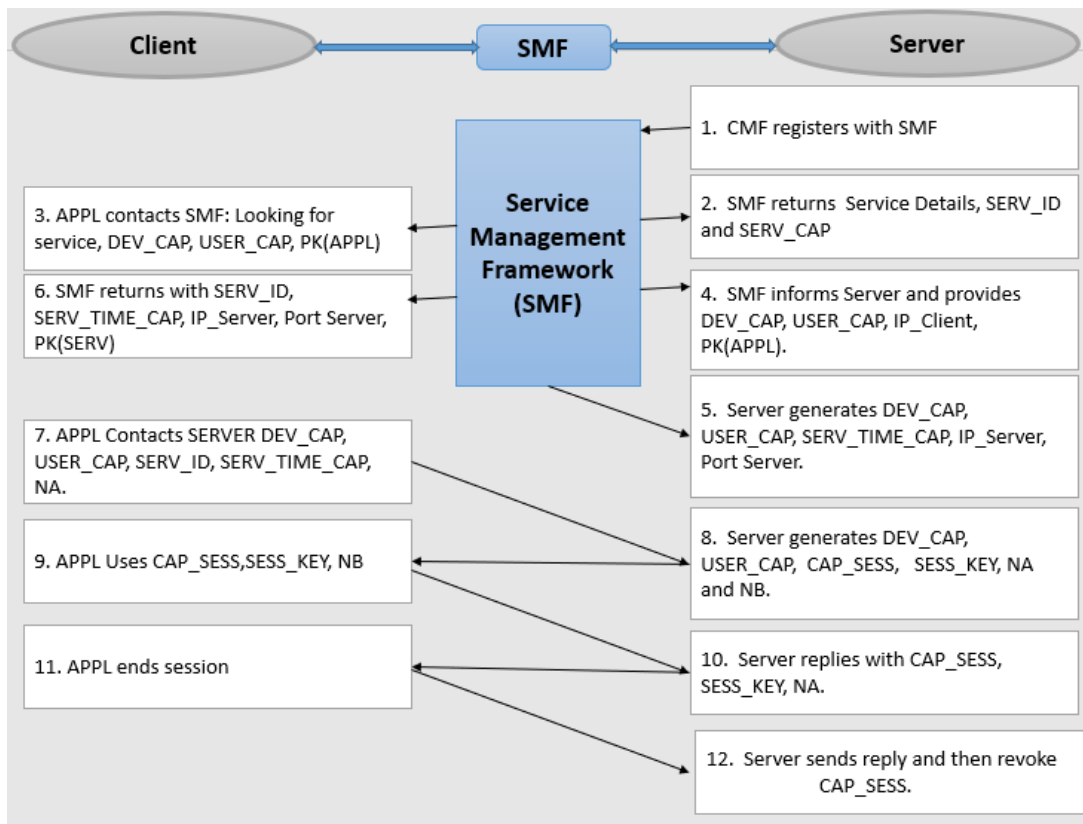


Figure 6.5: Secure Service Protocol

- Stage 1: Registration of the Service with the SMF** In the first step, the Cloud Management System (CMS), which supports numerous services, contacts the SMF to register the Service, including the list of servers, with their public keys running the Service. In step 2, the SMF registers the Service and returns the Service ID (SERV ID) and the Service Capability (SERV CAP). The CMS then starts the servers implementing the Service on different machines, passing Service ID and the Service Capability.
 1. CMS → SMF: (Register Service (Service Details), (PK(SMF)))
 2. SMF → CMS: (Service Registered (Service Details, SERV ID, SERV CAP), (PK(CMS)))
- Stage 2: The Application interacts with the SMF to get a server that implements the Service.** In order to use a service, an application must send a request to the SMF to find an appropriate server. In step 3, the Application passes the Capability for the Device (DEV CAP) and a Capability for the User of the Application (USER CAP), Service Name,

Service Version, and the Application's public key running on the machine. In step 4, the SMF chooses a server of that Service and contacts the server using its public key, passing the Client's DEV CAP, USER CAP, and IP address. In the next step, the Chosen Server accepts the request and then takes the service capability and generates a timed capability, which will expire in a few/60 seconds if the Application does not connect to the server. This SERV TIME CAP is a proxy capability derived from the Service Capability. The SERV TIME CAP cannot be changed and is also a private capability, so only that USER CAP and DEV CAP can use the capability. The Chosen Server also passes the server's IP address, the TCP port of the Service, using the public key of the SMF. In the last step of Stage 2, the SMF now returns to the Application with DEV CAP, USER CAP, timed capability, IP address of Server, Port number of the services, and public keys of Chosen Server, allowing the Application to connect to the server.

3. APPL → SMF: (Request service (Service Name, Service Version, DEV CAP, USER CAP, PK(APPL)), (PK(SMF)))
4. SMF Chosen -_j Server: (Usage request (DEV CAP, USER CAP, IP Client), (PK(SERV))).
5. Chosen Server-_j SMF : SMF (Usage request accepted (DEV CAP, USER CAP, SERV TIME CAP, IP Server, Port Server), (PK(SMF)))
6. SMF → APPL: (Request service accepted (DEV CAP, USER CAP, SERV TIME CAP, IP Server, Port Server, PK(SERV)), (PK(APPL)))

- **Stage 3: The Application sets up a secure session with the Server.**

The Application sends a session start request to the Chosen Server with DEV CAP, USER CAP, SERV ID, Nonce of NA, the public key of Server and uses the SERV TIME CAP. Nonces are randomly generated numbers that can be used just once to ensure that old communication cannot be repeated/reprocessed. Nonce (NA) ensures that the session is unique on the Application side. In Step 8, the Server replies with a session capability (CAP SESS), DEV CAP, USER CAP, NA, NB and a session key (SESS KEY). SESS KEY is a symmetric key that is only used once for encrypting the details in one communication session. Nonce (NB) is used to ensure the session is unique on the Server side.

7. APPL → Chosen Server: (Session Start Request (DEV CAP, USER CAP, SERV ID, SERV TIME CAP, NA)), (PK(SERV)).
8. Chosen Server → APPL : (Session Start Request Accepted DEV CAP, USER CAP, CAP SESS, SESS KEY, NA, NB), (PK(APPL)).

-
- **Stage 4: The Application gets service by using the CAP SESS key and encrypts using the SESS KEY** In Stage 4, the Application sends a service request with CAP SESS, Nonce (NB) and uses the SESS KEY to the Chosen Server. The Server sends CAP SESS, NA, Result in Details and uses the SESS KEY to the Application. Here, NA is used to ensure that the session is unique on the Application side.

9. APPL → Chosen Server : (Service Request (CAP SESS, NB, Request Details)(SESS KEY))

10. Chosen Server → APPL: (Service Request Done (CAP SESS, NA, Result Details) (SESS KEY))

- **Stage 5: The Application is finished and terminates the session** In Stage 5, the Application sends end requests to Chosen Server with CAP SESS, Nonce (NB), Session End and uses the session key. The Server terminates the session and then revokes the CAP SESS capability to prevent replay attacks.

11. APPL → Chosen Server :(Session End Request Accepted (CAP SESS, Nonce (A), Session End)(SESS KEY)).

6.4.0.1 General Notations_SSP Algorithm

Table 6.1: General Notations

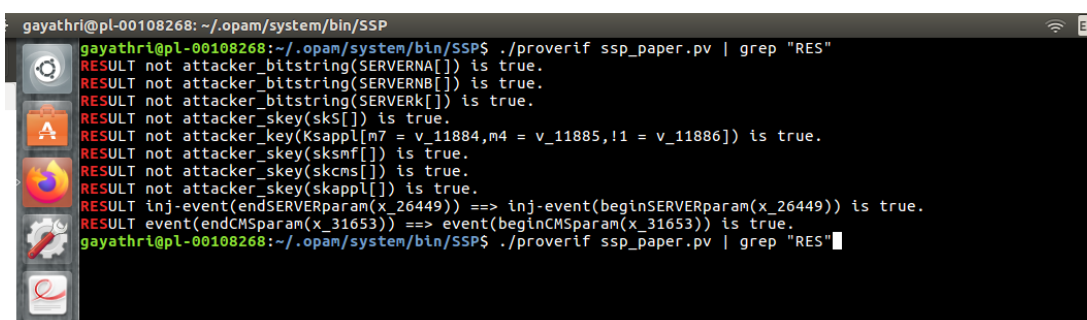
Notations	Explanation
CMS	Cloud Management System
SMF	Service Management Framework
APPL	Applications
Chosen Server	The SMF chooses a server of that service
PK(SMF)/PK(APPL)/PK(SERV)	Public key of SMF/APPL/Server
SK(SMF)/SK(APPL)/SK(SERV)/	Private key of SMF/APPL/Server
SESS KEY	Session key
NA and NB	Nonce of NA and NB
Register Service	CMS contacts to Register the Service
Service Details	Servers, keys and running services.
Service Registered	The SMF registers the Service
SERV ID	Service ID
SERV CAP	Service Capability
DEV CAP	The Capability for the Device
Request service	Request to an appropriate server
USER CAP	User of the Application
CAP SESS	User of the Application
Service Name	Name of the services
Service Version	Service Version
IP Client	IP address of Client
SERV TIME CAP	proxy capability
IP Server	IP address of Server
Port Server	TCP Port of Service
Usage request	SMF contacts the server
Usage request accepted	The Server accepts the request
Request service accepted	SMF returns all the details
Session Start Request	Time capability sends start request
Service request	The application requests service
Service request done	The application gets service
Request Details	Details of APPL request.
Result Details	Details of chose server result.
Session End Request	72 The application terminates the session
Session End Req accepted	Server terminates session

6.5 ProVerif results

The output of the query `attacker()` function results in "True", which means the attacker cannot be able to gain the value of that security property. The tool proves the attacker's knowledge and also protects the secrecy of data by authenticating between parties or more roles to securely transfer data and authenticated communications for an unbounded number of sessions using unbound data. If the outcomes of the Query Attacker () of function is "False", the value of the security property is accessible by the attacker. Finally, the result shows the attack trace if any or display query attacker of TRUE or FALSE. The SSP protocol is designed using ProVerif tool, and it results as "safe". In Stage 4, the application gets service by using the capability session key and encrypts using the Session key to the Chosen server. The query attacker of function to "Session key" (`Ksappl`) is TRUE.

The private keys of Cloud Management System (CMS), Service Management Framework (SMF), Application (Appl), Chosen Server(S) results are True. In Stage 3 and Stage 4, Nonces are used between the chosen Server and the Application to prevent the replay attacks. The output of Nonce is "True" from Server sessions. The Cloud Management System (CMS) which supports a number of services contacts the SMF to register the Service. We applied "event begin and end function" to verify whether the sessions started before the SMF starts or not and it results as "True". In the same way, `inj-begin` and `inj-end` function used to verify from Server Sessions.

The below Figure 6.6 and Table 6.2 show that the SSP protocol is working fine. Hence, this is a more secure mechanism for capability-based services.



```
gayathri@pl-00108268: ~/opam/system/bin/SSP
gayathri@pl-00108268:~/opam/system/bin/SSP$ ./proverif ssp_paper.pv | grep "RES"
RESULT not attacker_bitstring(SERVERNA[]) is true.
RESULT not attacker_bitstring(SERVERNB[]) is true.
RESULT not attacker_bitstring(SERVERK[]) is true.
RESULT not attacker_skey(skS[]) is true.
RESULT not attacker_key(Ksappl[m7 = v_11884,m4 = v_11885,!1 = v_11886]) is true.
RESULT not attacker_skey(sksmf[]) is true.
RESULT not attacker_skey(skcms[]) is true.
RESULT not attacker_skey(ksappl[]) is true.
RESULT inj-event(endSERVERparam(x_26449)) ==> inj-event(beginSERVERparam(x_26449)) is true.
RESULT event(endCMSparam(x_31653)) ==> event(beginCMSparam(x_31653)) is true.
gayathri@pl-00108268:~/opam/system/bin/SSP$ ./proverif ssp_paper.pv | grep "RES"
```

Figure 6.6: SSP protocols Results

6.5.1 Queries for Private keys

Queries for Private keys of Application, Cloud Management System, Service Management Framework and Server : `query attacker(sksmf)`; `query attacker(skcms)`;

Table 6.2: QUERY ATTACKER() RESULTS

Security Properties	CMF	SMF	SERVER	APPL
Private Keys	True	True	True	True
Session Key	-	-	True	True
Nonces	-	-	True	-
Event Begin()	True	-	-	-
Event end()	True	-	-	-
inj-event begin()	-	-	True	-
inj-event-end()	-	-	True	-

query attacker(skappl); query attacker(skS);

- RESULT not *attacker_skey(sksmf[])* is true.
- RESULT not *attacker_skey(skcms[])* is true.
- RESULT not *attacker_skey(skappl[])* is true.
- RESULT not *attacker_skey(skS[])* is true.

6.5.1.1 Queries for nonces

Queries for nonce between Server and application: *query attacker(SERVERNA); attacker(SERVERNB); attacker(APPLNB); attacker(APPLNA);*

- RESULT not *attacker_bitstring_(SERVERNA[])* is true.
- RESULT not *attacker_bitstring_(SERVERNB[])* is true.

6.5.1.2 Queries for Symmetric key

Queries for Symmetric key : *query attacker(Ksappl); query attacker(SERVERk)*

- RESULT not *attacker_key (Ksappl[m7 = v_10131,m4 = v_10132,!1 = v_10133])* is true.
- RESULT not *attacker_bitstring (SERVERk[[]])* is true.

6.5.1.3 Queries for authentication of Server event and CMS event

Queries for Symmetric key : *query attacker(Ksappl); query attacker(SERVERk)*

- RESULT inj-event(endSERVERparam(x_24696)) → inj-event(beginSERVERparam(x_24696)) is true.
- RESULT inj-event(endCMSparam(x_38772)) → inj-event(beginCMSparam(x_38772)) is true.
- RESULT event(endCMSparam(x_42655)) → event(beginCMSparam(x_42655)) is true.

6.6 Chapter Summary

So far, in this work, a new Resource Allocation Security Protocol (RASP) has been proposed for server migration between commercial Cloud environments. In our first attempt, using AVISPA, we showed that the protocol is safe under normal operation; the present protocol critically prevents impersonation attacks either by rogue Cloud infrastructure hoping to sneer valid services or by malicious servers wanting to inflict damage on Cloud Infrastructure. In our second attempt, using ProVerif, we showed that the proposed protocol succeeds in three significant security properties namely: secrecy, authentication of SA to CB and CB to SA, and secure symmetric key exchange. In this chapter, we started by showing the scenario diagram of Service Management Framework, Capabilities and then discussed the SSP protocol and showed the protocol is safe using ProVerif. This research has suggested a new Secure Service Protocol, which can be used to protect any service, using new techniques like Capabilities (used to identify any object and its properties) and a new Service Management Framework. The protocol has been verified using Proverif and is being deployed on the Middlesex VANET Testbed.

Chapter 7

Implementation

7.1 Brief Introduction

This chapter identifies the main contributions of the research, which will focus on developing the Service Management Prototype in the proposed security framework. In chapter 5, we showed the implementation of the RASP protocol and messages are exchanged to ensure security. In chapter 6, we extended this concept by adding capabilities and replacing the registries with the Service Management Framework (SMF). In this chapter, we seek to implement a real secure service transfer mechanism using migration techniques such as Docker, Capabilities and Service Management Framework (SMF).

7.2 Basic capability System Library (BCSL)

To build our prototype system, we used the Basic Capability System Library (BCSL), which is the Base Layer of the Implementation Framework, as our starting point. This layer supports users, devices, and services and allows servers to be added to the services. Each of them has its own capability, which is used to access other objects, for example, files. Services are created and are then registered with the Service Management Layer (SManL). Figure 6.4 in Chapter 6 discusses clearly how SMF works. The middle layer is used for enhancement, tuning and mapping layers or updates. The Basic Capability System Library was written by the ALERT Team and is only a basic system that enables the creation of users, devices, services and servers and implements them. These types are given as CAP USER, CAP DEVICE, and CAP SERVICE as their Capability types, respectively. The top layer is system implementation. Here, we are trying to improve the SMF to allow it to migrate and securely replicate the services. Our part of the work mainly focuses on how to migrate and replicate the registered

services securely.

Implementation Framework

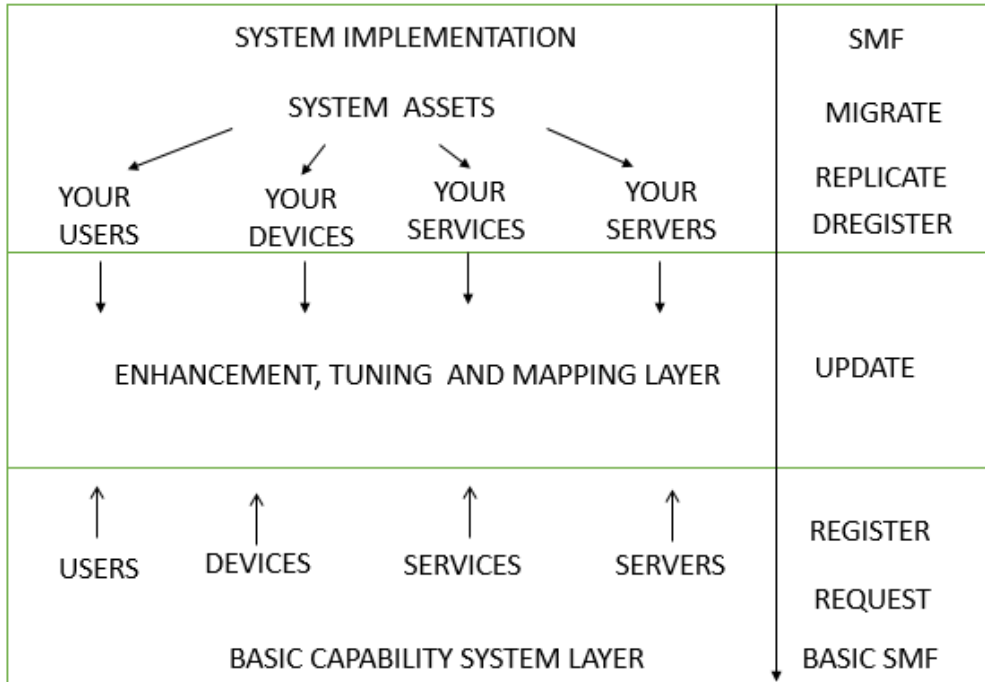


Figure 7.1: Basic capability System Library (BCSL)

7.3 Filesystem structure

Since we are looking at migrating the NMS, which is being used as the backend of the FUSE system, it is important to understand how filesystems work. In an operating system, the filesystem structure is the most basic level of organisation. The way an operating system organises data on storage devices affects almost every aspect of how it interacts with its users, programmes, and security model. The central concepts are superblock, inode, data block. Each file is represented by an inode. Each inode has several blocks.

- Super Block: This block contains administrative information about the partition's filesystem such as its size, dependencies information.

-
- inode: An inode contains the entire information about a file. The file name will be stored in the directory with the no of the inode. The inode contains several data blocks which allow the data to be stored in a file.
 - data blocks: The actual data are stored in the files on the filesystem. The data blocks are allocated to file systems dynamically as they are required.

7.3.1 Data blocks

The below struct structure shows how blocks are allocated locally. And then, allocated on the NMS if permanent storage is required.

```

struct block id
{
    unsigned int local blockid;    /*local block id*/
    unsigned int global block id; /*global block id*/
    struct capability raw;        /*capability blocks for NMS*/
    unsigned int status; /*status of the block on client*/
    unsigned char *data; /* the actual buffer */
};

```

- BIT(0): This status shows that the blocks is in memory.
- BIT(1): This status shows that the blocks has no data.
- BIT(2): This status shows that the blocks has been modified/altered.
- BIT(3): This status shows that the blocks has been allocated globally.

7.3.2 The structure of CAP FILE

The status field for type field(8 bits) as follows:

- Typefield (8 bits): File

The sys field (4 bits) as follows:

- BIT(P): If set, this file is private → Set
- BIT(S): this file is created by the system → Set
- BIT(M): If set, this file holds a master capability → Set
- BIT(C): If set, this file capability is changeable → Set

The status field for Property field (12 bits) is defined as follows:

- BIT(0): If set, Read → Set
- BIT(1): Read/Write → Set
- BIT(2): Execute → Set
- BIT(3): Delete → Set

The Object_Id is given as:

- BIT(0): If set, the file is a directory.
- BIT(1): If set, the file is an executable file.
- 32-bit: IP address of the device that manages the file.
- 28-bit: inode number of the file on that device

Random and Hash defined as:

- Random bit field (16 bits)
- Hash field (16 bits)

7.3.3 inode

The inode data structure keeps all the information files such as directories, files, codes, programs and everything on the server. Linux must allocate index inode for each file and directories in the file system.

```
struct ux inode {
    char filename [UX NAMELEN]; /* filename */
    unsigned int filenamelen; /* length of file name */
    struct capability raw mcap; /* the master capability */
    struct capability raw rwcap; /* the read write capability */
    struct capability raw rocap; /* the read only capability */
    struct capability raw xcap; /* the execute capability */
    struct file id fid; /* file object structure */
    int i type;
    /*type:inherit from parent:public,group,etc */
    int i oaccess; /*access allowed by other groups */
    int i inode access; /*whether inode access is allowed */
    int i inode access res; /*restrictions to inode access */
}
```

```

struct ux directory *i di; /*if the file is a directory */
uint32 t i no; /*inode number */
uint32 t i par no; /* the parent inode */
uint32 t i status; /* the status bits */
uint32 t i mode; /* whether file or directory */
uint32 t i nlink; /* number of links to storage systems*/
uint32 t i atime; /* last time accessed */
uint32 t i mtime; /* last time modified */
uint32 t i ctime; /* time when status changed */
int32 t i uid; /* uid of owner */
int32 t i gid; /* group owner of file */
uint32 t i blocks; /*the no of blocks in the file */
int32 t b offset /*the no of bytes in the last blocks */
int32 t f size; /*the size of the file in bytes */
uint32 t i addr [UX DIRECT BLOCKS];
/* the block id of the individual
blocks */
struct ux inode *next; /* inode list */
struct ux inode *prev; /* inode list */
};

```

7.3.4 RPC

Remote procedure Call (RPC) protocol is a set of procedures in a filesystem. It helps to read and write the files. This RPC allows a program on one system to remotely execute the program at another address in a network. Additionally, the messages between the NMS and the filesystem all use the same message format as detailed below:

```

struct nms rpc {
    unsigned short command;
    unsigned short reply;
    unsigned int local block id;
    unsigned int clnt ip address;
    unsigned int global block id;
    struct capability raw block
    char data[1024];
};

```

- unsigned short command: GET BLOCK, READ BLOCK, WRITE BLOCK, DELETE BLOCK.

-
- unsigned short reply refers the reply to command from the NMS
 - unsigned int local block id: this is the block on the client machine.
 - unsigned int clnt ip address: the IP address on the client machine
 - unsigned int global block id: the block on the NMS struct capability raw block capr.
 - struct capability raw block: the capability for the block on NMS

7.4 NMS working with the normal FUSE filesystem

The Fuse and NMS server is implemented for storing and managing the data residing in the system. As we stated already in Chapter 3, NMS provides networked block storage. If a client moves to another network, then the NMS is migrated to the new network to provide better performance. The Figure 7.2 shows client "fuxfs" and server "fuxfs_server" under the normal communication.



Figure 7.2: Normal NMS operation

7.4.1 Fuxfs server

The NMS server is a simple program: so should be easy to migrate between networks. This NMS server is working with the FUSE system. The header file for this program is named "fuxf.h" and executable code is in "Fuxfs_server". To compile and run the Fuse server.

- Compile command: `gcc -o Fuxfs_server Fuxfs_server.c`
- Run command: `./Fuxfs_server 0.0.0.0`

7.4.2 Fuxfs client

This is the client side of the Fuse code.

- Compile command: `gcc -Wall fuxfsmg.c 'pkg-config fuse --cflags --libs' -o fuxfs -w -L./ libcapab.a -lpthread`
- Run command: `./Fuxfs /tmp/fuse 0.0.0.0`

7.4.3 Make Fuxfs Multithreaded

In order to develop the prototype, it was decided to make the client (i.e the fuxfs) multithreaded. One thread called `SMF_THREAD` managed the interaction with the SMF as shown in Figure 7.3.

- `Make_user`
- `Make_device`
- `Make_service`
- `Add_server`
- `Request_service`
- `Migrate_service`

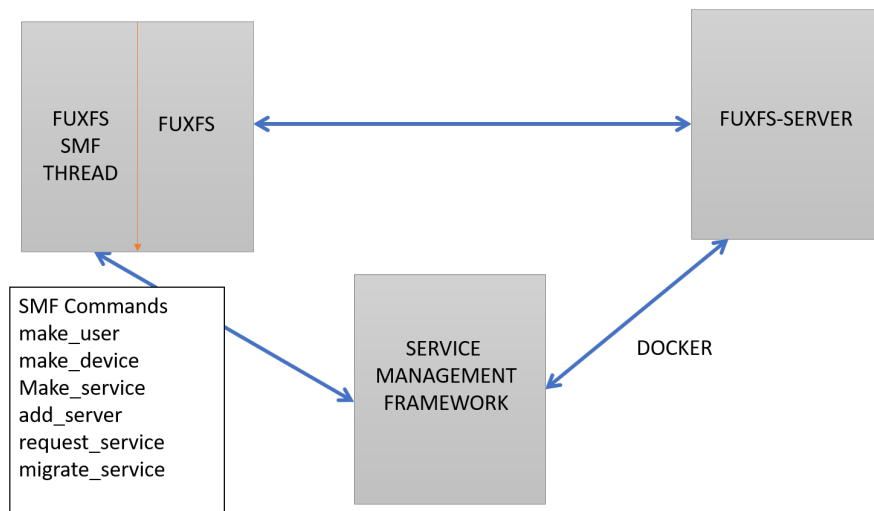


Figure 7.3: Fuxfs Multithread

7.5 SMF

The SMF was developed to manage the services, migrate and replicate the services in a secure environment. To check the secure migration on Service Management Framework (SMF), we developed C programs on the Ubuntu 20.04 platform, used a private Docker repository to migrate our service and tested them in our local environment. The SMF provides the following functions.

7.5.1 Register Service

The function allows the service to be registered with the SMF. It collects information on the service including service name, the description of service, the version of service (i.e., 0 is the latest version), the TCP/UDP port number, and the service capability. Once the service has been successfully registered with the SMF, the SMF generates the `global_id`. The `local_id` is generated by the client.

7.5.2 Register User

This function helps the new user to be registered to the SMF. It collects information about the user, including the first name, surname, role, rank, specialist, as well as the capability of the user to register. Once the user has been successfully registered with the SMF, the SMF will generate the global user ID for them. Each device will have a unique local device ID (generated by the `SMF_THREAD`) and a global device ID (generated by the SMF).

7.5.3 Register Device

This register device function allows the devices to register with SMF. To register a device including the name of a device, the first name and surname of the device owner, and the device capability is also collected once the device is registered. Each device will have a unique local device ID and a global device ID.

7.5.4 Add Server

This function helps the server be added to the required service. It also collects details about the server, including the server global ID given by NMS, the server's name, the server location using the IPv4 address, and the maximum load of the server.

7.5.5 Request Service

Once a service is registered with the SMF, the service can be requested by users who want to use it. The Servers should provide this service to the requested applications. This function allows an application to request service. After the application sends a service request to SManL, SManL will provide the necessary parameters for the application to contact the server that runs the requested service.

7.5.6 Migrate Service

This function allows the services to migrate to the required servers. It also collects details about global service ID, the global user_id, and the global device_id and will set up the migration using Docker container technology. Once the migration is successful, a successful result will be returned to the caller.

7.6 Testing and Evaluation

7.6.1 Docker in detail

Docker is a container technology, lightweight and easy to migrate applications and services from one location to another. Docker hub or docker repository has public, private and container repositories. This Docker hub has a collection of docker images. In our implementation, we use Docker as a migration mechanism to migrate the service. Please refer appendix for more details.

7.6.2 Docker Hub

In our research, we created an account in the Docker hub. Once we logged in, we created a private repository to store our services and the computing resources to enable migration in Cloud environments.

7.6.3 SMF - FUSE as a Service

We want to use the framework to implement microservices in vehicular environments. Microservices are faster to migrate and hence can be used to maintain QoS in these networks. As our first use case, we will consider a FUSE file system that is a user space file system commonly employed in Linux OS. In order to test the Service Management Framework, Fuse service will be migrated from one Cloud server to another. In my research, we show that by implementing a fuse file system in Docker's private repository and using it at a new location.

7.6.4 Steps for FUSE Server Migration in Docker Mechanism

The FUSE migration using a Docker container can be described by the following steps.

- The Fuse server is checked whether it's compiling and executing as expected in the source host. As per the Dockers procedure, Now, the Fuxfs_server container is pushed successfully to the private repository. We need to write a "Dockerfile" within the same folder to provide instructions such as OS name, version of the application, pathname, path location, compilation and execution commands or all the commands a user could call on the command line to assemble an image.
- The Fuse container's current status is pushed to Docker's private repository as an image. We should ensure that we are logged into the Docker hub.
- If we should call the newly created image to run on a different location or target machines, using the run command, you can convert the images as a container to run on the targeted machine.

7.7 FUSE Server Migration in Docker - Results

7.7.1 Source code compilation in Command Prompt

In Command prompt, compile the Fuse server code using the code below to check whether the source code is working. The program is called fuxfs_server.c in the fuxfs-master folder.

- `gcc fuxfs_server.c -o fuxfs_server`
- `./fuxfs_server`

7.7.2 Docker build - Fuse server

To run Docker, build your Docker image and check the built image is created or not. The below Figure shows [7.4](#) and [7.5](#).

- `docker build -t gayuinfy/gkprivate/fuxfs-server:v1 .`
- `docker images`

```

gk419@pl-00108268: ~/Desktop/fusemigration/fuxfs-master
Successfully built 536ec667b681
Successfully tagged gayuinfy/gkprivate/fuxfs-server:v1
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker build -t gayuinfy/gkprivate/fuxfs-server:v1 .
Sending build context to Docker daemon 117.8kB
Step 1/11 : FROM debian as build-dev
--> 04fbdaf87a6a
Step 2/11 : LABEL version="v1"
--> Using cache
--> a96cb0b7935d
Step 3/11 : RUN apt-get update && apt-get install -y g++ rsync zip openssl-server make
--> Using cache
--> c795a4d6ac05
Step 4/11 : WORKDIR /app
--> Using cache
--> 9f87b04b81fe
Step 5/11 : COPY . .
--> Using cache
--> 7b5e460f94ef
Step 6/11 : RUN gcc -o fuxfs_server fuxfs_server.c
--> Using cache
--> 241d8ca5f712
Step 7/11 : FROM debian
--> 04fbdaf87a6a
Step 8/11 : COPY --from=build-dev /app/fuxfs_server /app/fuxfs_server
--> Using cache
--> 32cc6e1b551b
Step 9/11 : WORKDIR /app
--> Using cache
--> 14f4b96e1c19
Step 10/11 : EXPOSE 1068
--> Using cache
--> 3d9cd56acc5f
Step 11/11 : CMD ["/app/fuxfs_server"]
--> Using cache
--> 536ec667b681
Successfully built 536ec667b681
Successfully tagged gayuinfy/gkprivate/fuxfs-server:v1
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$

```

Figure 7.4: Fuse Server - Build

```

gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
gayuinfy/gkprivate/fuxfs-server  v1          536ec667b681  4 minutes ago  124MB
<none>              <none>      241d8ca5f712  4 minutes ago  415MB
gayuinfy/dockerdocument  latest      4870e4fb059d  4 days ago    1.37GB
gayuinfy/myprg-gayu_test  0          4870e4fb059d  4 days ago    1.37GB
gayuinfy/myprg-gayu_test  1.1        4870e4fb059d  4 days ago    1.37GB
dockerdocument          latest      4870e4fb059d  4 days ago    1.37GB
myprg-gayu              latest      4870e4fb059d  4 days ago    1.37GB
postgres               9.6        027ccf656dc1  6 days ago    200MB
mongo                  latest      5285cb69ea55  2 weeks ago   698MB
ubuntu                 latest      54c9d81cbb44  2 weeks ago   72.8MB
redis                  latest      f1b6973564e9  3 weeks ago   113MB
debian                 latest      04fbdaf87a6a  3 weeks ago   124MB
hello-world            latest      feb5d9fea6a5  4 months ago  13.3kB
redis                  4.0        191c4017dcdd  22 months ago 89.3MB
postgres               10.10     9a05a2b9e69f  2 years ago   211MB
gayuinfy/myfuse         latest     6fa4d39cd2ba  3 years ago   85.9MB
gcc                    4.9       1b3de68a7ff8  4 years ago   1.37GB
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$

```

Figure 7.5: Docker image for Fuse Server

7.7.3 Create private repository in Docker Hub

To access the Docker hub, log in with your account, click create a repository on the first page, as we are working on a secure service migration, we have created a private repository and named it "gayuinfy/gkprivate". The below Figure shows 7.6 and 7.7.

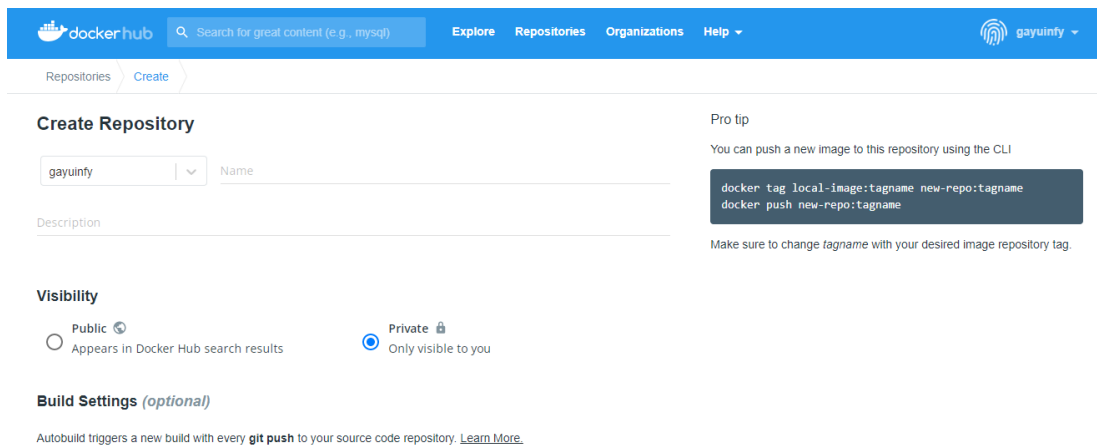


Figure 7.6: Docker Hub - Private repository Created

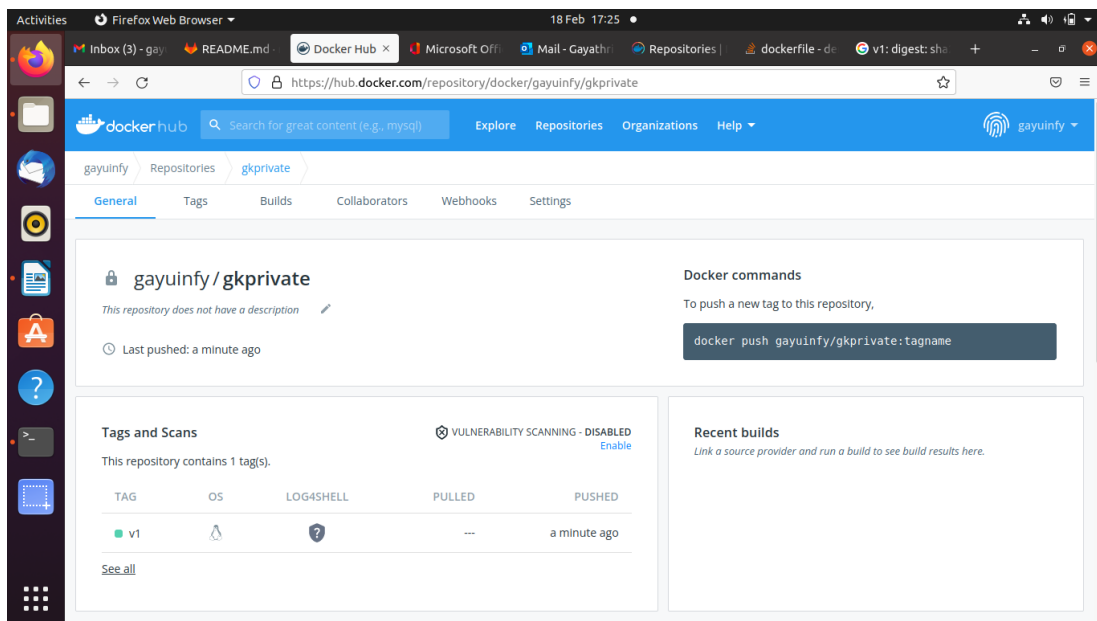


Figure 7.7: Docker Hub - Private repository

7.7.4 Push a Fuse container image to Docker Hub

To move the local image, from a local machine to Docker hub private repository, we need to run the below command. This Figures shows the pull and run results 7.8 and 7.9.

- docker login
- docker tag 536ec667b681 gayuinfy/gkprivate:v1

```

gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                NAMES
48f7022963dd  536ec667b681                       "/app/fuxfs_server"    7 minutes ago Up 7 minutes    1068/tcp            trusting_sinoussi
f7fc7a9ca254  536ec667b681                       "/app/fuxfs_server"    9 minutes ago Up 9 minutes     1068/tcp            unruffled_black
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker pull gayuinfy/gkprivate
Using default tag: latest
Error response from daemon: manifest for gayuinfy/gkprivate:latest not found: manifest unknown: manifest unknown
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker pull gayuinfy/gkprivate:v1
v1: Pulling from gayuinfy/gkprivate
Digest: sha256:d096ff6894576f06b7a948344e8ccd6089569f53db74a71e9b76965d44cb2323
Status: Image is up to date for gayuinfy/gkprivate:v1
docker.io/gayuinfy/gkprivate:v1
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker run gayuinfy/gkprivate:v1

```

Figure 7.8: Fuse Container pushed to Docker hub

```

Activities Terminal 18 Feb 17:24
gk419@pl-00108268: ~/Desktop/fusemigration/fuxfs-master
Push an image or a repository to a registry
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker images
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
gayuinfy/gkprivate/fuxfs-server  v1          536ec667b681  31 minutes ago  124MB
fuxfs-server        v1          536ec667b681  31 minutes ago  124MB
<none>              <none>      241d8ca5f712  31 minutes ago  415MB
gayuinfy/dockerdocument  latest      4870e4fb059d  4 days ago     1.37GB
gayuinfy/myprg-gayu_test  0           4870e4fb059d  4 days ago     1.37GB
gayuinfy/myprg-gayu_test  1.1        4870e4fb059d  4 days ago     1.37GB
dockerdocument        latest      4870e4fb059d  4 days ago     1.37GB
myprg-gayu           latest      4870e4fb059d  4 days ago     1.37GB
postgres             9.6        027ccf656dc1  6 days ago     200MB
mongo                latest     5285cb69ea55  2 weeks ago    698MB
ubuntu               latest     54c9d81cbb44  2 weeks ago    72.8MB
redis                latest     f1b6973564e9  3 weeks ago    113MB
debian               latest     04fbdaf87a6a  3 weeks ago    124MB
hello-world          latest     feb5d9fea6a5  4 months ago   13.3kB
redis                4.0       191c4017dcdd  22 months ago  89.3MB
postgres             10.10     9a05a2b9e69f  2 years ago    211MB
gayuinfy/myfuse      latest     6fa4d39cd2ba  3 years ago    85.9MB
gcc                  4.9       1b3de68a7ff8  4 years ago    1.37GB
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker tag 536ec667b681 gayuinfy/gkprivate:v1
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker login docker.io
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /home/gk419/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker push gayuinfy/gkprivate:v1
The push refers to repository [docker.io/gayuinfy/gkprivate]
db7cf3e78a92: Pushed
0bb0f2f5279: Pushed
v1: digest: sha256:d096ff6894576f06b7a948344e8ccd6089569f53db74a71e9b76965d44cb2323 size: 737
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                NAMES
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                NAMES

```

Figure 7.9: Fuse Container pushed to Docker hub

- docker login docker.io
- docker push gayuinfy/gkprivate:v1

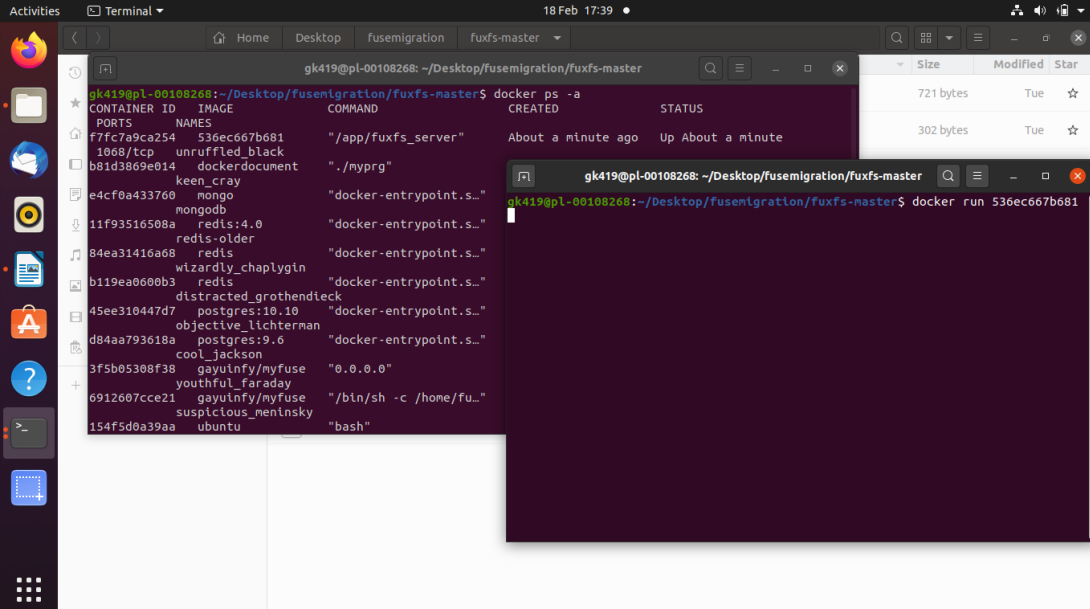
Now, the Fuse container is successfully pushed to the private repository. We can pull this images to test our Docker image locally.

- docker pull gayuinfy/gkprivate:v1
- docker tag 536ec667b681 gayuinfy/gkprivate:v1
- docker login docker.io
- docker push gayuinfy/gkprivate:v1

7.7.5 Run image from Docker Hub

To run a Fuxfs_server image inside of a Docker container, we used the docker run command 7.10.

- docker run gayuinfy/gkprivate:v1
- docker run 536ec667b681
- docker docker ps -a



```
gk419@pl-00108268: ~/Desktop/fusemigration/fuxfs-master
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
f7fc7a9ca254   536ec667b681                       "/app/fuxfs_server"    About a minute ago    Up About a minute
1068/tcp       unruffled_black                    "/bin/sh"              About a minute ago    Up About a minute
b81d3869e014   dockerdocument                    "./myprg"              About a minute ago    Up About a minute
e4cf0a433760   keen_cray                           "docker-entrypoint.s..." About a minute ago    Up About a minute
mongodb
11f93516508a   redis:4.0                            "docker-entrypoint.s..." About a minute ago    Up About a minute
redis-older
84ea31416a68   redis                                "docker-entrypoint.s..." About a minute ago    Up About a minute
wizardly_chaplygin
b119ea660b3    redis                                "docker-entrypoint.s..." About a minute ago    Up About a minute
distracted_grothendieck
45ee310447d7   postgres:10.10                       "docker-entrypoint.s..." About a minute ago    Up About a minute
objective_lichterman
d84aa793618a   postgres:9.6                          "docker-entrypoint.s..." About a minute ago    Up About a minute
cool_jackson
3f5b05308f38   gayuinfy/myfuse                       "0.0.0.0"              About a minute ago    Up About a minute
youthful_faraday
6912607cce21   gayuinfy/myfuse                       "/bin/sh -c /home/fu..." About a minute ago    Up About a minute
suspicious_mentinsky
154f5d0a39aa   ubuntu                                "bash"                 About a minute ago    Up About a minute

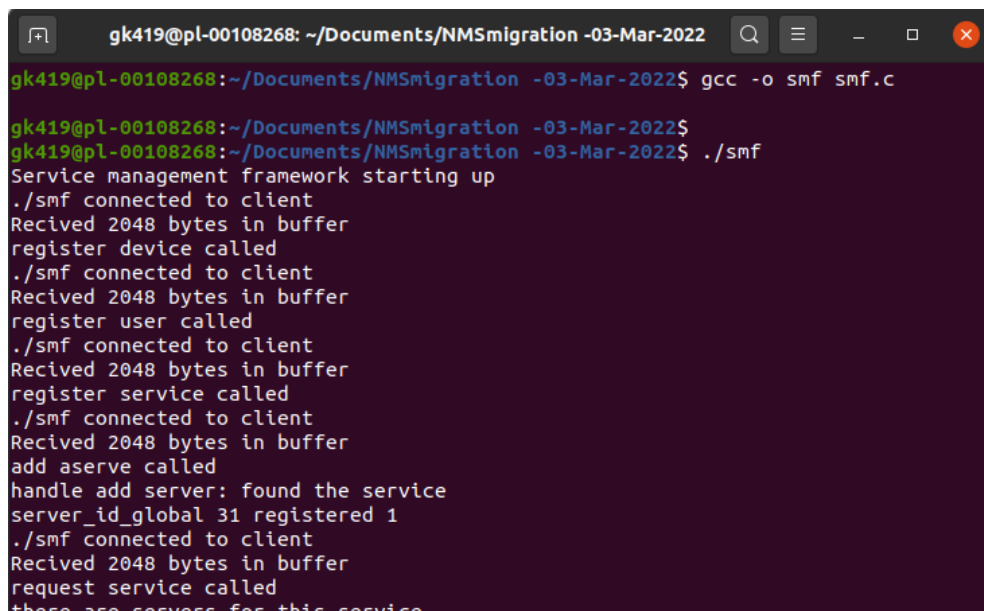
gk419@pl-00108268:~/Desktop/fusemigration/fuxfs-master$ docker run 536ec667b681
```

Figure 7.10: Run image from Docker Hub

7.7.6 Starting the Service Management Framework

The smf executable file has the functions to register service, register user, register device, add server, request and migrate the service. It was compiled and executed to check whether it can be able to start the framework using the below command. Figure 7.11 shows that SMF has successfully started. Once the fuse client is started, the SMF will show the state of ./smf connected to the client.

- Compile command: `gcc -o smf smf.c`
- Run command: `./smf`

A terminal window screenshot showing the execution of the SMF framework. The terminal title is "gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022". The user enters the command `gcc -o smf smf.c` and then `./smf`. The output shows the framework starting up and receiving multiple client connections. The output text is:

```
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ gcc -o smf smf.c
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./smf
Service management framework starting up
./smf connected to client
Recived 2048 bytes in buffer
register device called
./smf connected to client
Recived 2048 bytes in buffer
register user called
./smf connected to client
Recived 2048 bytes in buffer
register service called
./smf connected to client
Recived 2048 bytes in buffer
add aserve called
handle add server: found the service
server_id_global 31 registered 1
./smf connected to client
Recived 2048 bytes in buffer
request service called
there are servers for this service
```

Figure 7.11: SMF Execution

7.7.7 Fuxfs server start up

After SMF started, the Source code of "Fuxfs_server.c" compile and executed 7.12.

- Compile command: `gcc -o Fuxfs server Fuxfs server.c`
- Run command: `./Fuxfs server 0.0.0.0.`

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./fuxfs_server 0.0.0.0
I've received 8197 bytes from the network
buffer offset is 8197

Allocate block
I've allocated block number 1
I've sent 4 bytes over the network

I've received 0 bytes from the network
Socket was closed

I've received 8197 bytes from the network
buffer offset is 8197

Writing block 1

I've received 0 bytes from the network
Socket was closed

I've received 8197 bytes from the network
buffer offset is 8197

Writing block 1
```

Figure 7.12: Fuxfs server start up

7.7.8 Fuxfs client start up

After the Fuse server, the fuse client "Fuxfs.c" compiled and executed as shown in Figure 7.13. The SMF does not register a superuser because superusers have to do with local machines. But you have to be a superuser to make devices, make users, register devices, register service, add servers and migrate the service.

- Compile command : `gcc -Wall fuxfsmg.c `pkg-config fuse --cflags --libs` -o fuxfs -w -L./ libcapab.a -lpthread`
- Run command: `./fuxfs /tmp/fuse 0.0.0.0`

7.7.9 Making the Super user

The Super user has an administrative level access permission in the system. To create users, the fuse client system switches into superuser mode. Figure 7.14 shows how the account has been created for super users.

7.7.10 Make user

The "Make user" command allows users and their capabilities to create roles, specialists, etc. Figure 7.15 shows how the user account has been created for users and the user was also registered with SMF after being created.

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ ./fuxfs /tmp/fuse 0.0.0.0

Allocating block from the server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
I've received 4 bytes from the network
Received block number 1
Time spent in allocate_block is 380 us
Add entry . to inode 1

Writing block number 1 to server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
Time spent in write_block is 107 us
Add entry .. to inode 1

Writing block number 1 to server
Connected to the server 0.0.0.0
I've sent 8197 bytes over the network
Time spent in write_block is 94 us
Add entry lost+found to inode 1

Writing block number 1 to server
```

Figure 7.13: Fuxfs client start up

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Time spent in write_block is 72 us
SMT thread started
Making superuser
making superuser account: account name su
Please type your passwd: press enter when finished
1234
Please type your passwd again to confirm: press enter when finished
1234
Passwords match: Thank you
make_superuser: printing su capability
Superuser capability
The private is not set
The system is set
The master bit is set
The change bit is set
EUI number:6D7E8D9G
looking at: the netadmin field
value of the interface field: 0
object is stationary
the scope field is zero
the value of the random field: 4567
the value of the hash field: bb78
Made Supersuer
Please type su passwd: press enter when finished
```

Figure 7.14: Making superuser

7.7.11 Make device

The "Make device" command allows devices and their capabilities (Dev_cap) to be created including device name, owner of the device, EUI 8 byte code. Figure 7.16 shows how the device has been created and the device was also registered with SMF after being created.

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Thank you. Device successfully added
connected to the server
bytes received:2048 Result returned 0
we have interacted with the SMF
made device
making user
making user capability
Please type user surname: press enter when finished
karthick
karthick

Is this correct: Type y for YES and n for NO: press enter when finished
y
Please type user firstname: press enter when finished
Gayathri
Gayathri

Is this correct: Type y for YES and n for NO: press enter when finished
```

Figure 7.15: Make user

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Passwords match: Thank you
Making device
making device capability
Please type device name: press enter when finished
Laptop
Laptop

Is this correct: Type y for YES and n for NO: press enter when finished
y
Please type surname of owner of device: press enter when finished
karthick
karthick

Is this correct: Type y for YES and n for NO: press enter when finished
y
Please type firstname of owner of device: press enter when finished
Gayathri
Gayathri

Is this correct: Type y for YES and n for NO: press enter when finished
y
Please give the EUI of your device which is an 8-byte code
ED234YUI
ED234YUI
```

Figure 7.16: Make device

7.7.12 Make service

The "Make_service" command allows services to be created including service name, service description, binary of the service, password for the service and make sure this information is correct by asking Yes or No for each query. The Figure 7.17 shows how the service has been created and the service was also registered with SMF after being created.

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
bytes received: 2070 result returned 0
we have interacted with the SMF
Made user
Making service
making service capability
Please type service name: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
Please briefly describe the service in words use hyphens in between for example: block-s
storage-server: press enter when finished
block-storage-server
block-storage-server
Is this correct: Type y for YES and n for NO: press enter when finished
y
Please type where the binary for the service is located: press enter when finished
usr/bin/nms
usr/bin/nms
Is this correct: Type y for YES and n for NO: press enter when finished
y
Please type a passwd for the service: press enter when finished
1234
Please type your passwd again to confirm: press enter when finished
```

Figure 7.17: Make service

7.7.13 Add Server

The "Add server" command allows servers to add including Server name, Server Ip address, TCP and UDP port details and make sure this information is correct by asking Yes or No for each query. The Figure 7.18 shows how the Server has been added after the NMS service was found and the Server was also registered with SMF after being created.

7.7.14 Request service

The Request service command allows a user or application to request a service from the services registered with the SMF. The entity will be asked the name of the service and the version number. A Request Service command will be sent to the SMF who finds the service and returns all the details about the service including a server that is currently running the service as well as the global_id of the service which is used to migrate the service. Figure 7.19 shows that the NMS services was requested and that it was found on the SMF and the service structure including the NMS global_id was returned.

7.7.15 Migrate service

The "Migrate service" command allows servers to migrate and provide information including results, Global service id and Global Server id. The Figure 7.21 shows how the migration occurred and connected to the Server. The successfully

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
we have interacted with the SM
made service
Adding server
adding server to a service
Please type service name: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
service match found
add_server: found service:NMS
adding server to service: NMS
Please type name of the server: Type na if not applicable: press enter when finished
TG09
TG09
Is this correct: Type y for YES and n for NO: press enter when finished
y
add_server: please type the server location in terms of the IPv4 address in dot form: fo
r example 192.15.4.0. Press enter when finished
192.168.0.1
192.168.0.1
Is this correct? Type y for YES and n for NO

```

Figure 7.18: Add Server

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Is this correct? Type y for YES and n for NO
y
add_server: queued server on service
connected to the server
add_server: bytes received:2048 result returned 0
we have interacted with the SMF
Server added
Request Service starting up
connected to the server
Please enter name of service: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
Result returned: 2048
Service Registered:Global service id :11
Global server id: 31
Setting up migration
Migration set up and Ready to go
Migrating the Service

```

Figure 7.19: Request service

migrated service provides the same results, Global service id and Global Server id.

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Server 0000
Request Service starting up
connected to the server
Please type name of service: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
Result returned: 2048
Service Registered:Global service id :11
Global server id: 31

Setting up migration
Migration set up and Ready to go
Migrating the Service
Migrate Service starting up
connected to the server
Result returned: 2048
Service Registered:Global service id :11
Successfully Migrated the Service
Exiting..
```

Figure 7.20: Migrate service

```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
Server 0000
Request Service starting up
connected to the server
Please type name of service: press enter when finished
NMS
NMS
Is this correct: Type y for YES and n for NO: press enter when finished
y
Result returned: 2048
Service Registered:Global service id :11
Global server id: 31

Setting up migration
Migration set up and Ready to go
Migrating the Service
Migrate Service starting up
connected to the server
Result returned: 2048
Service Registered:Global service id :11
Successfully Migrated the Service
Exiting..

I'm in lookup!
```

Figure 7.21: Migrate service

7.7.16 Migrate service using Docker

The "Migrate service" uses a Docker container from one location to another to migrate the service. Figure 7.22 shows how the migration occurs using Docker and shows the status of the Fuse server starting in a new location.


```
gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
./smf connected to client
Recived 2048 bytes in buffer
register user called
./smf connected to client
Recived 2048 bytes in buffer
register service called
./smf connected to client
Recived 2048 bytes in buffer
add aserve called
handle add server: found the service
server_id_global 31 registered 1
./smf connected to client
Recived 2048 bytes in buffer
request service called
there are servers for this service
./smf connected to client
Recived 2048 bytes in buffer
request service called
Trying to migrate a program using fork and execPID of sample1.c = 9663
Parent process is running
Handle Migrate: We are ready to migrate
We are calling the migration program.
fuse server starting up
```

Figure 7.22: Migrate NMS service using Docker

7.7.17 Migrate service - Container Running Status

To check the status of Docker running from the Docker Hub private repository, we used the below command in a new terminal window. In SMF, we have set it up to call the function using `system()` command to call Fuse docker container using "docker run gayuinfy/gkprivate:v2 ". Figure 7.23 shows the Fuse running container status from the Docker hub name and private repository.

- `docker ps -a`

7.7.18 Final results of SMF

The Figure 7.24 shows the Fuse running container status from Docker hub name and private repository.

```

gk419@pl-00108268: ~/Documents/NMSmigration -03-Mar-2022
45ee310447d7 postgres:10.10 "docker-entrypoint.s..." 2 weeks ago
Exited (0) 2 weeks ago objective_lichterman
d84aa793618a postgres:9.6 "docker-entrypoint.s..." 2 weeks ago
Exited (1) 2 weeks ago cool_jackson
3f5b05308f38 gayuinfy/myfuse "0.0.0.0" 2 weeks ago
Created youthful_faraday
6912607cce21 gayuinfy/myfuse "/bin/sh -c /home/fu..." 2 weeks ago
Exited (255) 2 weeks ago suspicious_meninsky
154f5d0a39aa ubuntu "bash" 2 weeks ago
Exited (0) 2 weeks ago kind_goodall
e754cfbce357 myprg-gayu "./myprg" 2 weeks ago
Exited (0) 2 weeks ago confident_greider
1e0daa82deaf ubuntu "bash" 3 weeks ago
Exited (127) 3 weeks ago trusting_lovelace
13c9eadf7e10 hello-world "/hello" 3 weeks ago
Exited (0) 3 weeks ago zen_neumann
c1e67afba25e hello-world "/hello" 3 weeks ago
Exited (0) 3 weeks ago intelligent_chatelet
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$ docker ps
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS
ff0fa32d3036 gayuinfy/gkprivate:v2 "/app/fuxfs_server" 34 seconds ago Up
33 seconds 1068/tcp thirsty_ellis
gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022$

```

Figure 7.23: Docker Status: Fuse Container is running

```

gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022
g
g /smf connected to client
g Received 2048 bytes in buffer
g register user called
g /smf connected to client
g Received 2048 bytes in buffer
g register service called
g /smf connected to client
g Received 2048 bytes in buffer
g add_server called
g handle add servers: found the service
g server_id_global 31 registered 1
g /smf connected to client
g Received 2048 bytes in buffer
g request service called
g there are servers for this service
g /smf connected to client
g Received 2048 bytes in buffer
g request service called
g Trying to migrate a program using fork and execPid of sample.c = 9663
g parent process is running
g Handle Migrate: We are ready to migrate
g We are calling the migration program.
g fuse server starting up
g
g
g Is this correct? Type y for YES and n for NO
y
g add_server: please type the maximum load that the CPU is allowed to have as a percentage
g number, ie, out of 100. Press enter when finished
g 99
g
g Is this correct? Type y for YES and n for NO
y
g add_server: bytes received:2048 result returned 0
g we have interacted with the SMF
g Server added
g Request Service starting up
g connected to the server
g please type name of service: press enter when finished
g NMS
g Is this correct? Type y for YES and n for NO: press enter when finished
y
g Result returned: 2048
g Service Registered(global service id:11)

```

```

gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022
I've received 8197 bytes from the network
buffer offset is 8197
Allocate block
I've allocated block number 1
I've sent 4 bytes over the network
I've received 0 bytes from the network
socket was closed
I've received 8197 bytes from the network
buffer offset is 8197
Writing block 1
I've received 0 bytes from the network
socket was closed
I've received 8197 bytes from the network
buffer offset is 8197

```

```

gk419@pl-00108268:~/Documents/NMSmigration -03-Mar-2022
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS
ff0fa32d3036 gayuinfy/gkprivate:v2 "/app/fuxfs_server" 34 seconds ago Up
33 seconds 1068/tcp thirsty_ellis

```

Figure 7.24: SMF, FUSE Client and Server and Docker Status: Fuse Container is running

7.8 Chapter summary

In this chapter, we proposed our implementation framework including the Basic Capability System Library (BCSL) that was used to enhance the new Service Management Framework (SMF) to migrate services securely. This BCSL prototype was developed using the C programming language and preliminary performance results were obtained. We detailed the file system structure, Cap file structure, inode, and RPC. In order to test the system, we made the Fuxfs program multithreaded. This enabled it to interact with the SMF. Then, we migrated the NMS service using the Docker mechanism for secure service migration.

Chapter 8

Conclusion and Future Work

In this chapter, a summary of the accomplished work is presented and the major contributions to research and the wider field are highlighted. This is followed by conclusions derived from the work done and proposed future work for the research is presented.

8.1 Contribution of the Thesis

The contributions of the thesis can be summarised as follows:

- Chapter 1: In this thesis, we presented the Cloud computing overview, Cloud services, Cloud Advertisement in Commercial Environments and began by motivating the need for resource allocation applied in highly mobile environments, such as vehicular networks. It showed the secure protocol solution approach and key research questions that need to be addressed in this thesis.
- Chapter 2: A critical review of related literature on several areas of the existing solutions and approaches produced by researchers, scientists, and groups on Cloud computing environments like supporting highly mobile environments, Y-Comm reference framework, Cloud advertisements in a commercial environment, Service Oriented Architecture, Sardi's framework, MEC, Container technologies, resource allocation applied in Highly Mobile environment in Cloud., was presented in this chapter. In all these efforts reviewed, no resource allocation security protocol was developed and tested in a real-world environment for mobile users, which was the focus of this research.

-
- Chapter 3: This chapter described the significance of tools and technologies which have been used for conducting this research such as Service Migration by containers, VANET Clouds, Experiment Testbed, Use cases (FUSE file system and NMS) and formal verification tools (AVISPA and ProVerif).
 - Chapter 4: A detailed analysis of the Resource Allocation Algorithm for service migration between cloud systems and details of resources such as CPU, Memory, networking requirements, and storage was presented. In vehicular networks, the resource allocation algorithms need to be fast as users move around. Hence, a simple but effective algorithm was proposed and used in this research. This research helped focus on resource allocation security protocols for the safe migration of services between Cloud environments.
 - Chapter 5: In Chapter 5, a new Resource Allocation Security Protocol (RASP) was proposed. We verified the abstract protocol using two types of approaches: with AVISPA and with the ProVerif tool. In our first approach, this RASP protocol was verified using the AVISPA tool and proved to be safe for migration. The second approach, which uses session-based keys, has been verified using the ProVerif tool and was shown to be more secure to carry the service migration using session-based keys.
 - Chapter 6: In this chapter, new mechanisms such as the Capabilities based system and a New Service Management Framework (SMF) for mobile clouds or Vehicular Networks were proposed. In addition, a Secure Service Protocol (SSP) was introduced for secure service migration of any service. Then the SSP protocol was also verified using ProVerif.
 - Chapter 7: In this chapter, the Implementation Framework was presented, and several mechanisms were used, including the Basic Capability System Library (BCSL), NMS Normal Operation with FUSE, and the Multithreaded system that interacted with the SMF. The FUSE/NMS server was then migrated using Docker. The result of each step was presented in great detail and we showed that the proposed system was successfully tested.

8.2 Contributions to the research

This thesis has provided a comprehensive analysis of resource allocation security protocols for secure service migration that can be used to safely migrate services in Commercial Cloud environments. The key contributions of this thesis

are therefore as follows: A new Resource Allocation Security Protocol (RASP) has been proposed for the service migration between commercial Cloud environments. In our first attempt, using the AVISPA tool, we showed that the protocol is safe under normal operation; this protocol critically prevents impersonation attacks either by rogue Cloud infrastructure hoping to sneer valid services or by malicious servers wanting to inflict damage on Cloud Infrastructure. In our second attempt, using ProVerif, we showed that the proposed protocol succeeds in three significant security properties namely: secrecy, authentication of SA to CB and CB to SA, and secure symmetric key exchange. Furthermore, using new mechanisms such as Capabilities and a Service Management Framework, a Secure Service Protocol (SSP) was proposed for ANY service migration. SSP was verified using ProVerif and shown to be safe. Finally, a viable prototype was built using the Basic capability System Library(BCSL) to enhance the SMF, and a simple Network Memory Server (NMS) as part of the FUSE system was migrated between different servers using Docker mechanisms.

8.3 Contribution to the Field

This thesis considered resource allocation for secure Service migration in Cloud Environments that helps reduce operational costs, build a widely distributed development and deployment platform to access applications via the Internet, increase scalability, performance in local, national or global environments. We believe that this work can easily be integrated into current Cloud systems. This model can be applied to most communication systems in the real world.

8.4 Conclusion and Future Work

In this research, we focused on the verification of resource allocation protocols in the symbolic model using Applied pi-calculus. This thesis showed how the proposed protocols were verified with two well-known tools: AVISPA and ProVerif. To further validate the effort, a prototype was developed using the Basic capability System Library (BCSL), FUSE and NMS. The migration of the NMS using Docker mechanisms was clearly shown. We believe that this work can be further enhanced by using new and emerging technologies, such as Software Defined Vehicular Networking (SDVN), the Internet of things (IoT), 5G Internet, and Blockchain.

Bibliography

- Ocaml development-foreign function interface, 2019. URL <https://v1.realworldocaml.org/v1/en/html/foreign-function-interface.html>. 30
- Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *Journal of the ACM (JACM)*, 52(1):102–146, 2005. 28
- Mahdi Aiash, Glenford Mapp, Aboubaker Lasebae, Raphael Phan, and Jonathan Loo. A formally verified aka protocol for vertical handover in heterogeneous environments using casper/fdr. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):57, 2012. 30
- Rizwana Shaikh AR and Satish Devane. Formal verification of payment protocol using avispa. 2010. 25
- Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005. 26, 27
- Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012. 16
- M Bhaavan, KC Gouda, and Ananda KR Kumar. Virtualization, resource allocation and security measures in cloud computing. *International Journal of Science, Engineering and Computer Technology*, 4(6):190, 2014. 16
- Bruno Blanchet. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *Foundations of Security Analysis and Design VII*, pages 54–87. Springer, 2013. 30

- Bruno Blanchet et al. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1 (1-2):1–135, 2016. 16, 29
- Vincent Cheval Bruno Blanchet, Ben Smyth and Marc Sylvestre. Automatic cryptographic protocol verifier,user manual and tutorial. Book, November 2021. 28, 29
- Jichuan Chang, Parthasarathy Ranganathan, and Kevin T Lim. Memory server, February 21 2017. US Patent 9,575,889. 23
- S. Chen, H. Fu, and H. Miao. Formal verification of security protocols using spin. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6, June 2016. doi: 10.1109/ICIS.2016.7550830. 25
- Edmund M. Clarke and Bernd-Holger Schlingloff. Chapter 24 - model checking. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, Handbook of Automated Reasoning, pages 1635–1790. North-Holland, Amsterdam, 2001. ISBN 978-0-444-50813-3. doi: <https://doi.org/10.1016/B978-044450813-3/50026-6>. URL <https://www.sciencedirect.com/science/article/pii/B9780444508133500266>. 25
- H. Cloud. The nist definition of cloud computing. national institute of science and technology, special publication, 800(2011), p.145. 2011. 1
- Ashutosh Dutta and Eman Hammad. 5g security challenges and opportunities: A system approach. In *2020 IEEE 3rd 5G World Forum (5GWF)*, pages 109–114, 2020. doi: 10.1109/5GWF49715.2020.9221122. 16
- Thomas Erl. *SOA principles of service design (the Prentice Hall service-oriented computing series from Thomas Erl)*. Prentice Hall PTR, 2007. 12
- Onyekachukwu Augustine Ezenwigbo, Jose Ramirez, Gayathri Karthick, Glenford Mapp, and Ramona Trestian. Exploring the provision of reliable network storage in highly mobile environments. In *2020 13th International Conference on Communications (COMM)*, pages 255–260, 2020. doi: 10.1109/COMM48946.2020.9142033. 8
- Jipeng Gao and Gaoming Tang. Virtual machine placement strategy research. In *2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 294–297. IEEE, 2013. 14

- Dr Gayathri Karthick, Florian Kammüller, Glenford Mapp, and Mahdi Aiash. Exploring secure service migration in commercial cloud environments. In *WORKSHOP 2019*, page 19, 2019. 8
- Noha Hamdy, Amal Elsayed, Nahla ElHaggar, and M Mostafa-Sami. Resource allocation strategies in cloud computing: Overview. *International Journal of Computer Applications*, 975:8887, 2017. 16
- Zhenqiu Huang, Kwei-Jay Lin, and Chi-Sheng Shih. Supporting edge intelligence in service-oriented smart iot applications. In *2016 IEEE International Conference on Computer and Information Technology (CIT)*, pages 492–499, 2016. doi: 10.1109/CIT.2016.40. 10
- Julio A Hurtado Alegría, María Cecilia Bastarrica, and Alexandre Bergel. Avispa: a tool for analyzing software process models. *Journal of software: Evolution and Process*, 26(4):434–450, 2014. 27
- Rasheed Hussain, Junggab Son, Hasoo Eun, Sangjin Kim, and Heekuck Oh. Rethinking vehicular communications: Merging vanet with cloud computing. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 606–609. IEEE, 2012. 4
- G. Karthick, G. Mapp, F. Kammüller, and M. Aiash. Formalization and analysis of a resource allocation security protocol for secure service migration. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 207–212, Dec 2018. doi: 10.1109/UCC-Companion.2018.00058. 8, 60
- Gayathri Karthick, Glenford E Mapp, Florian Kammüller, and Mahdi Aiash. Exploring a security protocol for secure service migration in commercial cloud environments. 2017. 5, 8
- Gayathri Karthick, Glenford Mapp, Florian Kammüller, and Mahdi Aiash. Modeling and verifying a resource allocation algorithm for secure service migration for commercial cloud systems. *Computational Intelligence*, n/a(n/a), 2021. doi: <https://doi.org/10.1111/coin.12421>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12421>. 8
- Jun Kikuchi, Celimuge Wu, Yusheng Ji, and Tutomu Murase. Mobile edge computing based vm migration for qos improvement. In *2017 IEEE 6th global conference on consumer electronics (GCCE)*, pages 1–5. IEEE, 2017. 14
- Pascal Lafourcade, Vanessa Terrade, and Sylvain Vigier. Comparison of cryptographic verification tools dealing with algebraic properties. In *International*

- Workshop on Formal Aspects in Security and Trust*, pages 173–185. Springer, 2009. [30](#)
- Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. Mobile edge computing: Progress and challenges. In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pages 83–84. IEEE, 2016. [14](#)
- M. How avispa tool validates security protocols and applications. Web, 2022. [28](#)
- G. Mapp, F. Sardis, and J. Crowcroft. Developing an implementation framework for the future internet using the y-comm architecture, sdn and nfv. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 43–47, June 2016. doi: 10.1109/NETSOFT.2016.7502440. [12](#)
- Glenford Mapp, Mahdi Aiash, Hélio Crestana Guardia, and Jon Crowcroft. Exploring multi-homing issues in heterogeneous environments. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, pages 690–695. IEEE, 2011. [63](#), [64](#)
- Glenford E Mapp, Fatema Shaikh, Jon Crowcroft, David Cottingham, and Javier Baliosian. Y-comm: a global architecture for heterogeneous networking. 2007. [4](#), [6](#), [11](#)
- Victor Medel, Omer Rana, Jose Angel Banares, and Unai Arronategui. Modelling performance and resource management in kubernetes. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 257–262, 2016. [18](#)
- Vishnu Vardhan Paranthaman, Arindam Ghosh, Glenford Mapp, Victor Inioiosa, Purav Shah, Huan X Nguyen, Orhan Gemikonakli, and Shahedur Rahman. Building a prototype vanet testbed to explore communication dynamics in highly mobile environments. In *International Conference on Testbeds and Research Infrastructures*, pages 81–90. Springer, 2016. [19](#)
- Vishnu Vardhan Paranthaman, Yonal Kirsal, Glenford Mapp, Purav Shah, and Huan X. Nguyen. Exploring a new proactive algorithm for resource management and its application to wireless mobile environments. In *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pages 539–542, 2017. doi: 10.1109/LCN.2017.86. [15](#)
- Vishnu Vardhan Paranthaman, Yonal Kirsal, Glenford Mapp, Purav Shah, and Huan X Nguyen. Exploiting resource contention in highly mobile environments

- and its application to vehicular ad-hoc networks. *IEEE Transactions on Vehicular Technology*, 68(4):3805–3819, 2019. 5, 15
- Shiva K. Pentyala. Emergency communication system with docker containers, osm and rsync. 2017. 4
- Jose Ramirez, Onyekachukwu Augustine Ezenwigbo, Gayathri Karthick, Ramona Trestian, and G. Mapp. A new service management framework for vehicular networks. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 162–164, 2020. doi: 10.1109/ICIN48450.2020.9059441. 8, 10
- Fragkiskos Sardis, Glenford Mapp, Jonathan Loo, Mahdi Aiash, and Alexey Vinel. On the investigation of cloud-based mobile media environments with service-populating and qos-aware mechanisms. *IEEE transactions on multimedia*, 15(4):769–777, 2013. 4, 6, 10, 13
- Anurag Shashwat and Deepak Kumar. A service identification model for service oriented architecture. In *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, pages 1–5. IEEE, 2017. 12
- Johan Tordsson, Rubén S Montero, Rafael Moreno-Vozmediano, and Ignacio M Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future generation computer systems*, 28(2): 358–367, 2012. 15
- Kifayat Ullah, Luz M Santos, João B Ribeiro, and Edson DS Moreira. Sadrp: A lightweight beaconing-based commercial services advertisement protocol for vehicular ad hoc network. In *International Conference on Ad-Hoc Networks and Wireless*, pages 279–293. Springer, 2016. 12
- Bharath Kumar Reddy Vangoor, Vasily Tarasov, and Erez Zadok. To {FUSE} or not to {FUSE}: Performance of user-space file systems. In *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*, pages 59–72, 2017. 23
- Nattaruedee Vithanwattana, Gayathri Karthick, Glenford Mapp, and Carlisle George. Exploring a new security framework for future healthcare systems. In *2021 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2021a. 66
- Nattaruedee Vithanwattana, Gayathri Karthick, Glenford E Mapp, and Carlisle George. Exploring a new security framework for future healthcare systems. *Conference*, 2021b. 9

BIBLIOGRAPHY

- Peipei Xiao, Yufeng Wang, Qun Jin, and Jianhua Ma. A privacy-preserving incentive scheme for advertisement dissemination in vehicular social networks. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1741–1746, 2016. doi: 10.1109/TrustCom.2016.0267. [12](#)
- Ke Zhang, Yuming Mao, Supeng Leng, Alexey Vinel, and Yan Zhang. Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 288–294. IEEE, 2016. [14](#)
- Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010. [2](#)

Appendix

.1 Docker installation

Firstly, we installed Docker on our machine by typing the below commands based on our machine configuration. My machine configuration is Linux Ubuntu 20.04. Once the Docker installation is completed, Docker will start automatically or by typing the start and stop command.

- `sudo apt install docker.io runc`
- `sudo systemctl status Docker.`
- `sudo systemctl start docker`

To find the versions for Docker Client and Server versions by type Docker version and to get server permission, use `chmod`.

- Docker Version
- `sudo chmod 666 /var/run/docker.sock`