12-14-2015 12:00 AM

Security Protocol Suite for Preventing Cloud-based Denial-of-Service Attacks

Marwan M. Darwish
The University of Western Ontario

Supervisor
Dr. Abdelkader Ouda
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

SECURITY PROTOCOL SUITE FOR PREVENTING CLOUD-BASED DENIAL-OF-SERVICE ATTACKS

(Thesis format: Monograph)

by

Marwan Darwish

Graduate Program in Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Cloud systems, also known as cloud services, are among the primary solutions of the information technology domain. Cloud services are accessed through an identity authentication process. These authentication processes have become increasingly vulnerable to adversaries who may perform denial-of-service (DoS) attacks to make cloud services inaccessible. Several strong authentication protocols have been employed to protect conventional network systems. Nevertheless, they can cause a DoS threat when implemented in the cloud-computing system. This is because the comprehensive verification process may exhaust the cloud resources and shut down cloud's services. This thesis proposes a novel cloud-based secure authentication (CSA) protocol suite that provides a smart authentication approach not only for verifying the users' identities but also for building a strong line of defense against the DoS attacks. CSA protocol suite offers two modules, CSAM-1 and CSAM-2. The decision of which module of CSA to be utilized depends on the deployment nature of the cloud computing.

CSAM-1 is designed to prevent external risks of DoS attacks in private and community cloud computing. CSAM-1 utilizes multiple techniques that include the client puzzle problem and utilization of unique encrypted text (UET). Therefore, these techniques can distinguish between a legitimate user's request and an attacker's attempt.

CSAM-2 is designed to prevent internal risks of DoS attacks in public and hybrid cloud computing. CSAM-2 combines an extended unique encrypted text (EUET) application, client puzzle problem, and deadlock avoidance algorithm to prevent DoS risks that occur from inside cloud computing systems. The authentication process in both modules is designed so that the cloud-based servers become footprint-free and fully able to detect the signs of DoS attacks.

The reliability and scalability of these two modules have been measured through a number of experiments using the GreenCloud simulation tool. The experiments' results have shown that the CSA protocol suite is practically applicable as a lightweight authentication protocol. These experiments have verified the ability of the CSA to protect the cloud-based system against DoS attacks with an acceptable mean time to failure while still having the spare capacity to handle a large number of user requests.

Keywords

Cloud Computing, Denial-of-Service (DoS) Attack, Network Security, Authentication Protocol.

Acknowledgments

First of all, I thank God for giving me the ability to achieve my goals. I would like to thank my lovely parents and my lovely family (my wife and my two daughters) for their support and encouragement in my life.

I would like to thank my supervisor Prof. Abdelkader Ouda for his guidance and advice during my Ph.D. program. His academic expertise helped me to improve my research skills. His support and motivation gave me the confidence and the strength to accomplish my goals.

I acknowledge my former supervisor Prof. Luiz Capretz for his guidance and advice during my Ph.D. program.

I would like to thank Dr. Arif Raza for his valuable comments and suggestions on my final version of the thesis.

I acknowledge King Abdulaziz University and the Cultural Bureau of Saudi Arabia in Canada for their financial support and for their advising during my Ph.D. studies.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Program Interface
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
HCF	Hop Count Filtering
HIP	Host Identity Protocol
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPSec	Internet Protocol Security
MIPS	Million Instructions Per Second
OSI	Open Systems Interconnection
PaaS	Platform as a Service
RAM	Random Access Memory
SaaS	Software as a Service
SVO	Syverson & Van Oorschot
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
VM	Virtual Machine
WPA2	Wi-Fi Protected Access II

Chapter 1

1 Introduction

Security and reliability are important in the cloud computing environment. This is especially true today because denial-of-service (DoS) attacks constitute one of the largest threats to Internet users and cloud computing services. DoS attacks target the resources of these services, thereby lowering their ability to provide optimum usage of the network infrastructure. Owing to the nature of cloud computing, the methodologies for preventing or mitigating DoS attacks are quite different compared to those used in conventional networks. This chapter discusses the motivation and objectives of this research. It additionally explains the research methodology and the main contributions. Finally, this chapter describes the structure of the thesis.

1.1 Research Motivation

Addressing DoS attacks for all service models in cloud systems is a major challenge owing to the difficulty of distinguishing an attacker's attempt from a legitimate user's request, even though the requests originate from different distributed machines. DoS attacks affect all cloud system service models—Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)—and can occur from outside or from inside the cloud environment.

External cloud-based DoS attacks are initiated from outside the cloud environment and target cloud-based services. These attacks occur from outside the cloud system and target SaaS and PaaS models. This type of attack affects the availability of services.

Internal cloud-based DoS attacks raise from the cloud system itself, such as within the PaaS or IaaS model. This attack type can occur in several ways. For example, the attacker may exploit the trial period of the cloud service of a vendor. Consequently, an authorized user within the cloud environment can internally launch a DoS attack on the victim's machine. Moreover, sharing infected virtual machine (VM) images can allow an attacker to control and utilize the infected VMs to execute an internal DoS attack on the targeted machine within the same cloud-computing system.

The case scenarios that demonstrate how internal and external DoS attacks affect cloud-computing systems are described as follows.

1. **Case Scenario-1 E-mall Company External DoS Attack.**

E-Mall Company, which provides a web-based e-commerce service to their visitors and members, decided to move to a cloud-based hosting service. They agreed to migrate their service to Whatchamacallit Elastic Compute Cloud (WEC2) and utilize the Whatchamacallit Elastic Block Store (WEBS) service for their data storage. However, three months after the implementation of the cloud service, the E-Mall service became unavailable to their members and visitors for more than 16 hours. Neither could E-Mall Company control the circumstances nor could they address the cause of the unavailability of the service. They contacted Whatchamacallit support and were informed, "that E-Mall Company had somehow exhausted all its available bandwidth." E-Mall Company notified the Whatchamacallit support representative that their records indicated that only 150 transactions had been processed and that only 1,050 users were online when the services terminated. After some time, E-Mall Company was intimated, "that their bandwidth was consumed by the flood of innumerable User Datagram Protocol (UDP) packets that were targeted at their IP." This attack caused approximately 16 hours of service unavailability. Consequently, the E-Mall Company incurred the following costs:

- Owing to the nature of cloud computing and based on the WEC2 service agreement, the E-Mall Company was responsible for the bandwidth usage consumed during the DoS attack. The average cost of the bandwidth usage was $0.10 per GB.

- The web-based e-commerce system was unavailable resulting in a loss of approximately $50,000 worth sale transactions.

2. **Case Scenario-2 E-mall Company Internal DoS Attack:**

The second scenario is an internal DoS attack that occurred at the E-Mall Company two days following the resolution of the first DoS attack. During this attack, the website

of E-Mall Company was unable to access the WEBS to store or retrieve data. The response they received was, “that multiple transactions were performed on the data storage system triggering an extended communication response time with the database.” E-Mall Company “observed that their logs indicated that database transactions occurred at average levels when compared to their transaction volumes over the previous three months.” Following an investigation by Whatchamacallit, “it was discovered that a flood of TCP SYN connection requests affected the data-storage service.” WEBS service is an internal service and is not related directly to the customers’ systems. Therefore, the second attack, which affected the data-storage system, occurred through another virtual machine in the WEC2 environment. Whatchamacallit explained “that the second DoS attack differed from the first attack in that it was an internal DoS attack. It was mentioned that an owner of another virtual server within the cloud environment caused seven other virtual machines to flood the WEBS service concurrently by sending forged requests. Therefore, the WEBS response time was extremely slow during the attack.” As a result, E-Mall Company reported the following losses:

- E-Mall Company lost approximately $6,000 worth sales transactions because of the unavailability of the data for two hours.

- E-Mall Company lost 1,000 customers because the customers were frustrated by the effects of the internal DoS attack. They were not able to browse the E-Mall services as easily as they could browse for the services of other competing firms.

Owing to the above challenges, defending cloud-computing systems against the different types of DoS attacks is an important research issue. The defense objective is to maintain cloud system availability and avoid the attacks’ overloaded resource usage, which can cause financial losses. Accordingly, this research strives to:

- Develop methods to effectively protect cloud-based systems against external DoS attacks using strong authentication protocols that are targeted to external DoS attacks.

- Develop an approach to effectively defend cloud-based systems against internal DoS attacks by employing strong authentication protocols that are targeted to internal DoS attacks.

In addition, this research analyzes and validates the proposed cloud-based authentication protocol suite against both external and internal DoS attacks. The full scope of research objectives are described in the following section.

1.2 Research Objectives

The main goal of this thesis is designing and developing a novel cloud-based authentication protocol suite to securely authenticate the cloud user and to prevent risks of external and internal DoS attacks. The research objectives established to achieve the research goal are as follows:

1. Investigate DoS attacks in conventional networks and then identify DoS attacks in cloud-computing systems. The tasks for achieving this objective are to:
    - Explore the risks of DoS attacks.
    - Study existing types of DoS attacks in conventional network systems.
    - Examine existing defense mechanisms against DoS attacks in conventional network systems.
    - Evaluate the ability of cloud-computing systems to detect DoS attacks.
    - Identify and categorize all possible types of DoS attacks in cloud-computing systems.
2. Investigate, propose, and validate a protocol suite that defends the cloud-based system against external DoS attacks. The tasks to achieve this objective are to:
    - Design a cloud-based authentication protocol to defend the system against external DoS attacks.
    - Analyze and validate the security and performance of the proposed cloud-based authentication protocol.
3. Investigate, develop, and validate a defender protocol suite against internal DoS attacks. The following tasks to accomplish this objective are to:
    - Develop a cloud-based authentication protocol to prevent internal DoS attacks.

- Analyze and validate the security and performance of the proposed cloud-based authentication protocol.

1.3 Research Methodology

This section describes the methodologies that are applied in this research to design and develop the cloud-based authentication protocol suite that defends against external and internal DoS attacks. The proposed cloud-based authentication protocol suite can not only detect internal DoS threats, but it can also defend the system against external DoS attackers. In addition, the developed protocol satisfies the following security requirements:

- Multilevel adaptive technique that determines the efforts of the protocol participants.
- Ability to identify a legitimate user's requests prior to heavy authentication.
- Footprint-free core protocol.
- Formal validation of the protocol.
- Experimental validation of the protocol.

1.3.1 Defender Protocol Suite Design

The identity authentication protocol is believed to have become increasingly susceptible to attackers who use DoS techniques. A cloud-based authentication protocol that securely authenticates a cloud user and effectively prevents DoS attacks on the cloud-computing system is needed. In this thesis, we propose a cloud-based DoS-resistance protocol suite that securely authenticates cloud users. This protocol is designed such that the cloud server is required to do lightweight computation work without the need to store any data during the authentication process. In addition, it requires the cloud user to do a process that is computationally expensive.

To realize the Meadows cost-based approach (Meadows, 2001), this protocol suite considers a "subset sum" problem as a "client puzzles" technique (Juels & Brainard, 1999) (Chapters 3 and 4). A "subset sum" problem is a kind of cryptographic knapsack problem, and it is not only a strong one-way function, but it also has a flexibility property to be adaptive (Salomaa, 1996). The complexity of the subset sum knapsack problem depends

on the size of the knapsack (the total number of its items, n) and on the number of items (the subset sum size, m) involved in puzzle solution. If the number of items n is small, then an exhaustive search for the solution is practical. Also, if the number of m is small compared to n, then a solution can be found in a reasonable time. Consequently, by adjusting the values of n and m, determining the difficultly level of the knapsack problem and hence the cost-based approach can be adaptively realized. Therefore, the proposed cloud-based DoS-resistance protocol suite has the ability to be configured such that the more sensitive the services requested are, the greater the computation cost required from the requester is. In other words, greater computation cost can be achieved by asking the requester (client) to perform an expensive operation, such as solving an expensive subset sum puzzle. The responder (cloud server) should conduct inexpensive to medium-cost operations, such as generating subset sum puzzle elements, checking the solution, or decryption operations.

A banker's (Dijkstra, 2002) algorithm is also implemented in the second module of the proposed protocol to control the allocation of services and to prevent the risk of an internal DoS attack on a specific service host or virtual machine (Chapter 4).

1.3.2 Security and Performance Validation

To validate the security requirements of an identity authentication protocol, it is important to analyze the proposed protocol via two different models. Therefore, the proposed authentication protocol is formally analyzed via Syverson & Van Oorschot (SVO) logic (P. F. Syverson & Van Oorschot, 1994) to validate its security requirements (Section 5.1.2). The proposed protocol is also validated via a cost-based model approach (Meadows, 2001) to ensure that it is invulnerable to DoS attacks (Section 5.1.1).

In addition, to validate the performance of the proposed protocol, experiments are conducted to assess the ability of the protocol to serve as a DoS defender tool. The dynamic programming algorithm is our tool to solve and assess the computational cost of both participants (client and cloud server) involved in the subset sum problem. We were able to experimentally determine different values of n and m that provide different levels of the

subset sum solution's complexity (Section 5.2.1). This makes our protocol adaptive enough to protect different levels of sensitive services.

The GreenCloud simulator (Kliazovich, Bouvry, & Khan, 2012) was used to validate the proposed protocol implementation in terms of their ability to prevent DoS attacks. A real Linux OS was configured to collect real data regarding the properties of the high-level computational processes of the proposed protocol. These data include the number of million instructions per second (MIPS), the input size, the output size, and storage size of each process. The collected data were used in the simulation as the input parameters for simulating the protocol's processes. Furthermore, a virtual machine (VM) was configured as a cloud server in the simulation in order to execute the computational processes of the protocol under extreme conditions. Then, VM data were collected from the simulation tool, including the VM load, VM memory load, and VM storage load (Section 5.2.3). Finally, the analysis results from the collected VM data were compared to the analysis results from a widely used authorization protocol in the Internet (Sections 5.2.2 and 5.2.3). The main contributions of this thesis are detailed in the following section.

1.4 Main Thesis Contributions

This thesis focuses on designing and developing a novel authentication protocol suite to securely authenticate the cloud user and to prevent external and internal risks of DoS attacks. Research contribution can be mainly summarized as follows:

- A taxonomy of existing DoS attacks and defenses on cloud-computing systems.
- An investigation and design of a defender authentication protocol suite against external cloud-based DoS attacks.
- An investigation and design of a defender authentication protocol suite against internal cloud-based DoS attacks. In addition to the performance validation of the proposed authentication protocol.
- A security validation of the proposed authentication protocol suite against cloud-based DoS attacks.
- A performance evaluation of a widely used cloud-based authorization framework in the Internet.

The research contributions of this thesis have been published in journals and conference proceedings in the areas of information security and cloud security. Therefore, these contributions have been peer-reviewed by experts in the field.

1.5 Thesis Structure

The thesis structure is outlined as follows:

- Chapter 2 provides an overview of cloud-computing technology and DoS attacks. In addition, it provides a literature review of existing cloud-based DoS attacks and defenses. Furthermore, the ability of a cloud-computing system to defend against DoS attacks using an existing authentication protocol is also provided in this chapter.

- Chapter 3 presents and discusses the proposed cloud-based secure authentication module 1 (CSAM-1) protocol suite that is used to defend against external DoS attacks.

- Chapter 4 presents and discusses the proposed cloud-based secure authentication module 2 (CSAM-2) protocol suite that is employed to defend against internal DoS attacks.

- Chapter 5 describes a security and performance validation of the proposed protocol. It provides formal verification of the protocol via SVO logic, as well as security validation using a cost-based model. In addition, it provides an analysis of a puzzle technique that is implemented in the proposed protocol suite. Furthermore, it provides an evaluation of a widely used existing authorization framework. Finally, it provides a performance validation of the proposed protocol suite.

- Chapter 6 summarizes the contributions of the thesis and outlines the future work.

Chapter 2

2 Background and Literature Review

This chapter overviews the cloud computing in general and the related DoS attacks. It presents an in-depth security analysis of DoS attacks in cloud computing. Finally, it presents a literature review of the existing authentication protocols on cloud computing and their ability to defend against DoS attacks.

2.1 Cloud Computing

Cloud computing is the utilization of hardware and software to provide services to end users over a network, such as the Internet (Mell & Grance, 2011). It includes a set of VMs that simulates physical computers and provides services, such as operating systems and applications. However, configuring virtualization in a cloud-computing environment is critical when deploying a cloud-computing system.

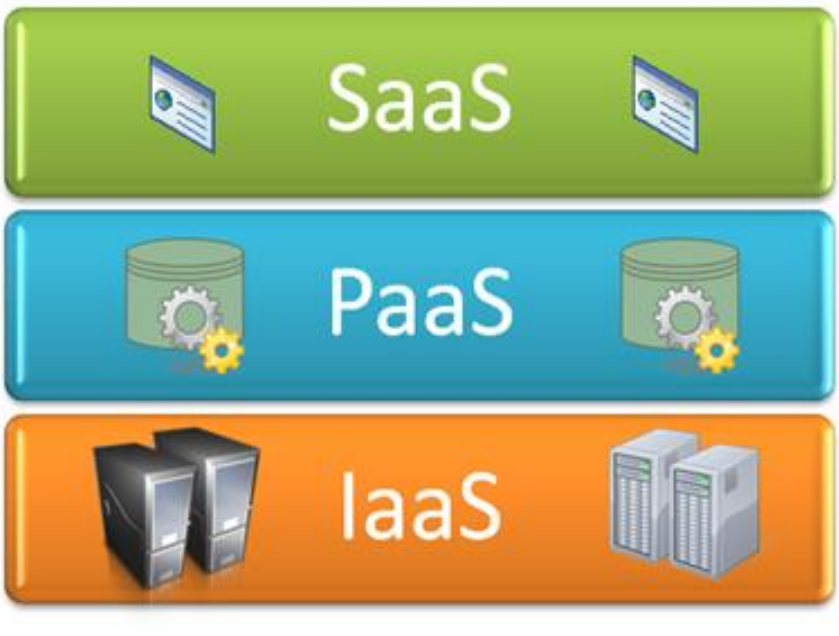

Figure 2.1: Architecture of cloud computing service models

A cloud-computing structure relies on three service models (Mell & Grance, 2011): IaaS, PaaS, and SaaS (Fig. 2.1). IaaS enables users to access physical resources, networks,

bandwidth, and storage. PaaS builds on IaaS and provides end users with access to the operating systems and platforms that are required for building and developing applications, such as databases. SaaS provides end users with access to software applications.

Furthermore, cloud computing can be implemented using different deployment models, as described below (Mell & Grance, 2011):

- **Private Cloud:** This cloud model is deployed to be used by a single organization. This deployment model could be owned, operated, and managed by the organization itself or by a third party exclusively for the organization. It can be located on or off the organization.

- **Community Cloud:** This cloud model is deployed to be used by multiple organizations that have shared interests such as government organizations. This deployment model could be owned, operated and managed by one or more organizations in the community or by a third party exclusively for the community. It can be located on or off the organizations.

- **Public Cloud:** This cloud model is deployed to be used by general public. This deployment model could be owned, operated, and managed by, for example, a business or an academic organization. It is located in a cloud provider.

- **Hybrid Cloud:** This deployment model is a combination of two or more different deployment models (such as private, community, or public) that communicate together to allow application and data portability. It is to be noted that the hybrid cloud in this research is considered as a combination of the public model and either private, community, or both (private and community) models.

2.2 Denial-of-Service Attacks

DoS attacks are a major security risk in the cloud-computing environment, where resources are shared by many users. A DoS attack targets resources or services to attempt to render them unavailable by overloading system resources with substantial amounts of spurious traffic (Mather, Kumaraswamy, & Latif, 2009). The objective of DoS attacks is to consume

critical resources, such as memory, CPU processing space, or network bandwidth, to make them unreachable to end users by blocking network communication or denying access to services.

DoS attacks are becoming increasingly more sophisticated. Many websites and large companies have been targeted by these attacks. The first DoS attack was reported in 1999 (Nazario, 2008). In 2000, large resource companies, such as Yahoo, Amazon, CNN.com, and eBay, were targeted by DoS attacks, and their services were stopped for hours (Neumann, 2000). Register.com was targeted by a DoS attack in 2001. It was the first DoS attack to use domain name servers (DNSs) as reflectors (Dittrich, Mirkovic, Reiher, & Dietrich, 2004). In 2002, a service disruption was reported at 9 of 13 DNS root servers on account of a DNS backbone DoS attack. This attack type reoccurred in 2007 and disrupted two DNS root servers (Arora, Kumar, & Sachdeva, 2011). In 2007, a DoS attack was executed by thousands of computers targeting more than 10,000 online game servers (Arora et al., 2011). In 2008, a DoS attack targeting Wordpress.com resulted in 15 minutes of service denials (Patel & Borisagar, 2012). In 2009, GoGrid, a cloud-computing provider, was targeted by a massive DoS attack that affected approximately half of the thousands of customers of the provider. In 2009, Register.com was again targeted by a DoS attack. In the same year, several social networking sites, including Facebook and Twitter, were targeted by various DoS attacks. Many websites were affected by DoS attacks in 2010, including the Australian Parliament House website, Optus, Web24, Vocus, and the website of Burma's main Internet provider. In 2011, Visa, MasterCard, PayPal, and PostFinance were targeted by a DoS attack that aimed to support the WikiLeaks founder (Patel & Borisagar, 2012). In the same year, the site of the National Election Commission of South Korea was targeted by a DoS attack.

Furthermore, thousands of infected computers were used in a DoS attack that targeted the Asian E-Commerce Company in 2011 (Patel & Borisagar, 2012). In 2012, the official website of the office of the vice-president of Russia was unavailable for 15 hours owing to a DoS attack (Patel & Borisagar, 2012). In the same year, many South Korean and United States (US) websites were targeted by DoS attacks. Godaddy.com websites reported service outages because of a DoS attack in the same year. In 2012, major US banks and

other financial institutions became targets of a DoS attack. As evidenced by the above cases, the volume of DoS attacks is rapidly increasing. Moreover, these attacks are targeting major companies, which are consequently incurring significant global financial losses.

DoS attacks, such as those above, include different types of techniques. These techniques and their effects are outlined below.

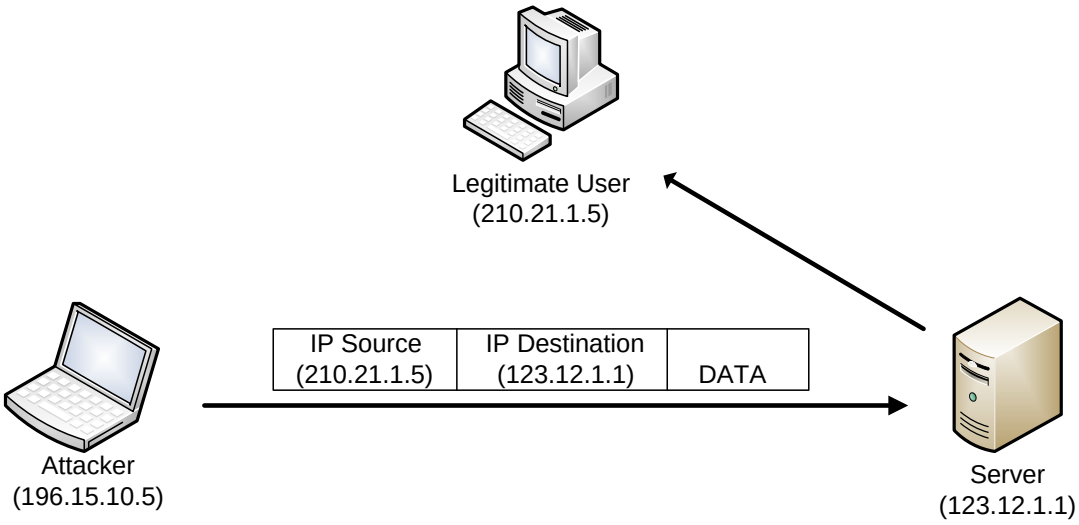

Figure 2.2: IP spoofing attack

- **IP spoofing attack:** In an Internet Protocol (IP) spoofing attack, packet transmissions between the end user and cloud server are intercepted. Their headers are modified so that the IP source field in the IP packet is forged by using either a legitimate IP address or an unreachable IP address, as shown in Fig. 2.2. Consequently, the server responds to the legitimate user machine, thereby affecting that machine, or the server is unable to complete the transaction to the unreachable IP address, which affects the server resources. Tracing such an attack is difficult because of the forged IP address in the IP source field of the IP packet.

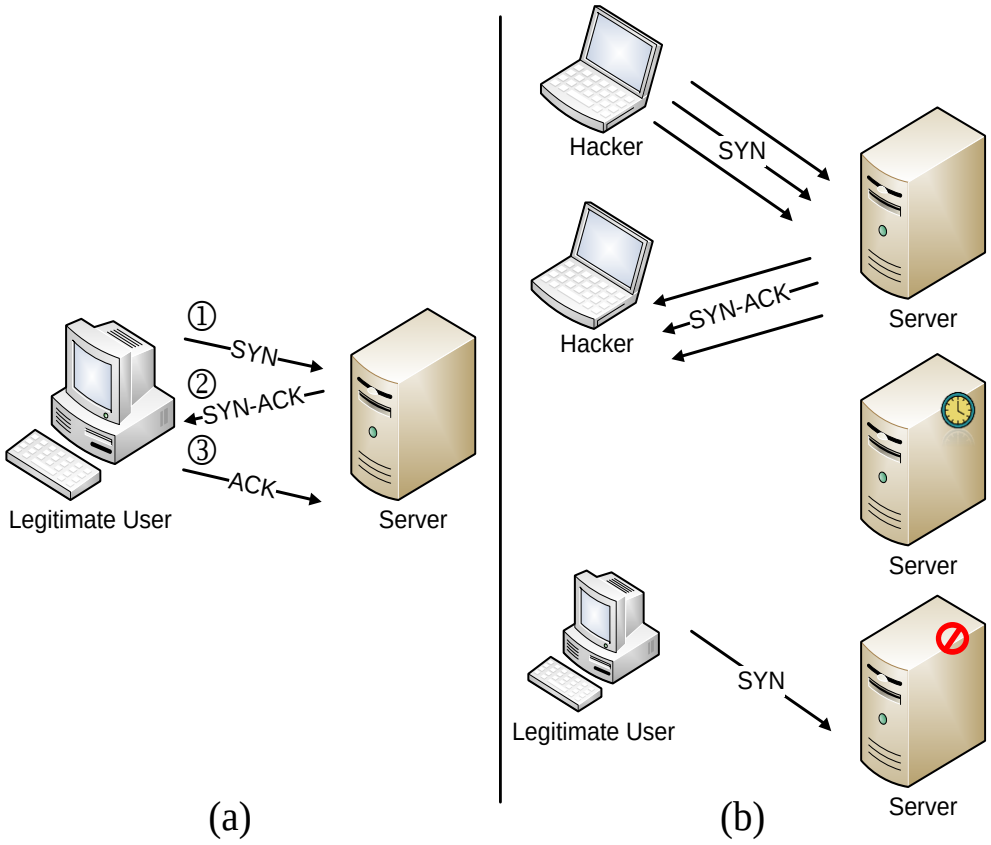

Figure 2.3: (a) Normal three-way handshake; (b) SYN flooding attack

- **SYN flooding attack:** A Transmission Control Protocol (TCP) connection starts with a three-way handshake, as shown in Fig. 2.3(a). A typical three-way handshake between a legitimate user and the server begins by sending a connection request from the legitimate user to the server in the form of a synchronization (SYN) message. Subsequently, the server acknowledges the SYN message by returning a request (SYN-ACK) to the legitimate user. Finally, the legitimate user sends an ACK request to the server to establish the connection. SYN flooding occurs when the attacker sends innumerable packets to the server but does not complete the three-way handshake process. Consequently, the server waits to complete the process for all of those packets. This prevents the server from processing legitimate requests, as shown in Fig. 2.3(b). Moreover, SYN flooding can be executed by sending packets with a spoofed IP address. A sniffing attack is considered a type of SYN flooding attack. In a sniffing

attack, the attacker sends a packet with the predicted sequence number of an active TCP connection with a spoofed IP address. Thus, the server is unable to reply to that request, thereby affecting the performance of the cloud system because of extensive resource consumption.

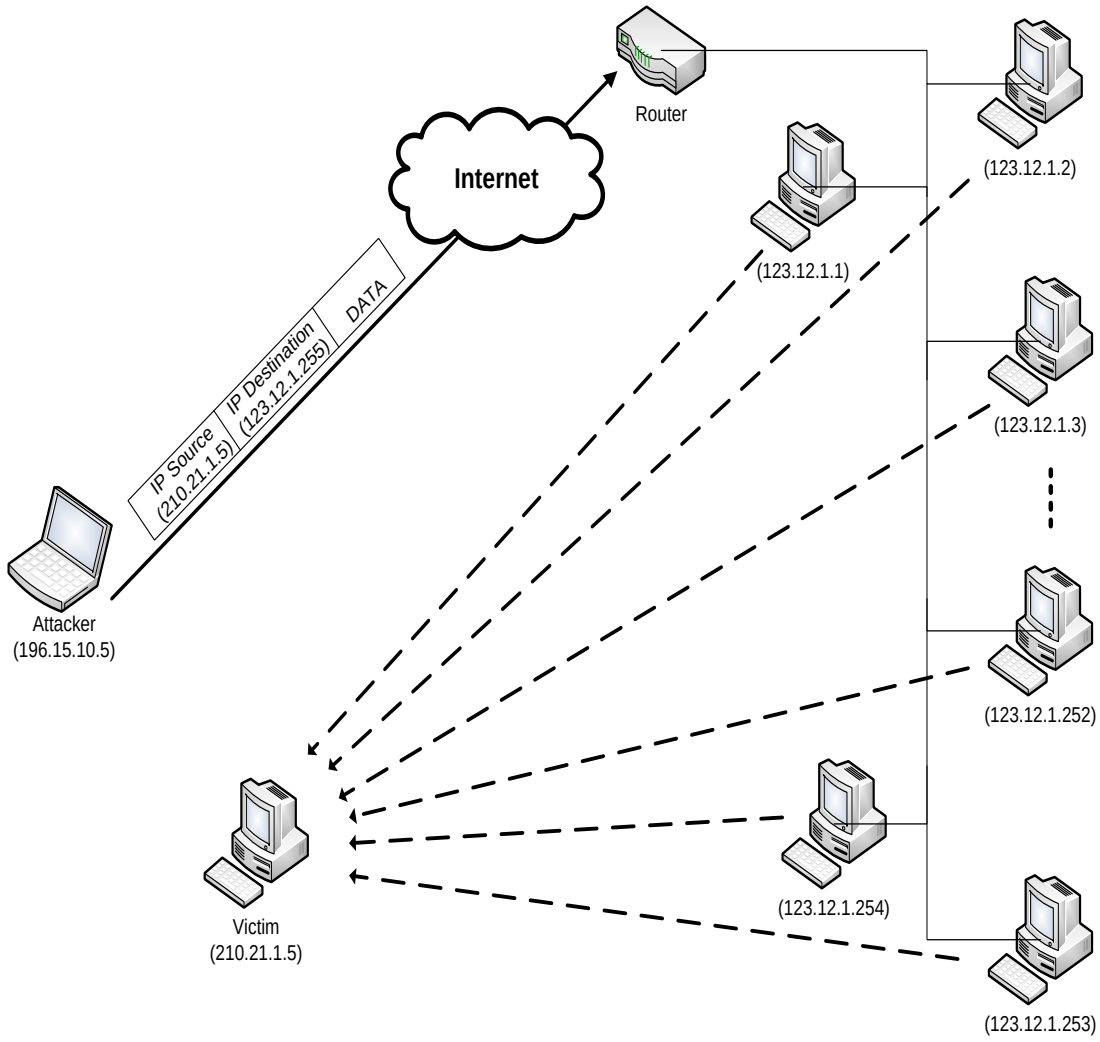

Figure 2.4: Smurf attack

- **Smurf attack:** In a smurf attack, the attacker sends many Internet Control Message Protocol (ICMP) echo requests. These requests are spoofed so that their source IP address is the IP address of the victim, and the IP destination address is the broadcast

IP address, as shown in Fig. 2.4. Therefore, the victim is flooded with broadcasted addresses. In the worst case scenario, the number of hosts who reply to the ICMP echo requests is excessively large.

- **Buffer-overflow attack:** In a buffer-overflow attack, an attacker sends executable code to the victim to exploit the buffer-overflow vulnerability. Consequently, the attacker can completely control the victim machine. The attacker can subsequently either harm that machine or use the infected machine to perform an internal cloud-based DoS attack.

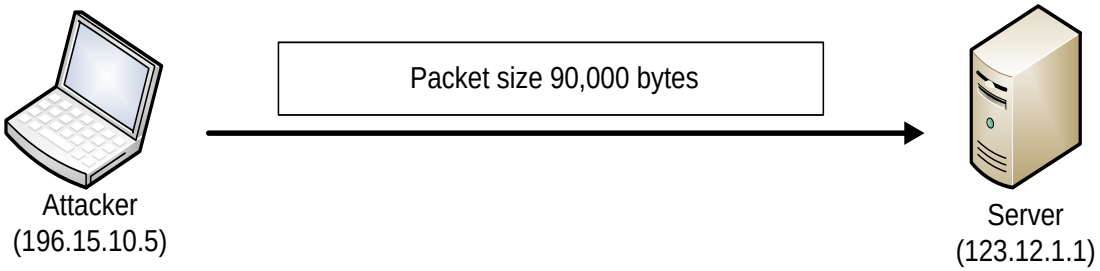

Figure 2.5: Ping of death attack

- **Ping of death attack:** In the ping of death attack, the attacker sends an IP packet larger than the IP protocol limit, which is 65,535 bytes, as shown in Fig. 2.5. Processing an oversized packet affects both the victim machine within the cloud system and the cloud system resources.

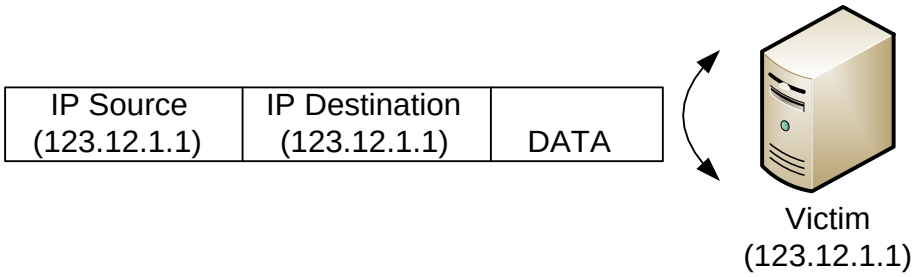

Figure 2.6: Land attack

- **Land attack:** This attack uses the Land.c program to send forged TCP SYN packets with the victim's IP address in the source and destination fields, as shown in Fig. 2.6. The machine receives the request from itself, which crashes the system.

- **Teardrop attack:** This type of attack uses the Teardrop.c program to send invalid overlapping values of IP fragments in the header of TCP packets. The victim machine in the cloud system will crash in the re-assembly process.

2.3 Literature Review

2.3.1 DoS Defense Techniques

DoS attacks in cloud-computing environments can be externally or internally initiated, as shown in Fig. 2.7. An external cloud-based DoS attack is launched from outside the cloud system and targets the cloud's services to disrupt their availability. Therefore, an external DoS attack can affect the SaaS and PaaS models. On the other hand, an internal cloud-based DoS attack originates from inside the cloud system, primarily in the IaaS and PaaS models. These attacks can take various forms. For example, an attacker can take advantage of the free trial periods of some cloud service providers. Hence, an authorized user of the cloud system may internally initiate a DoS attack on the targeted services.

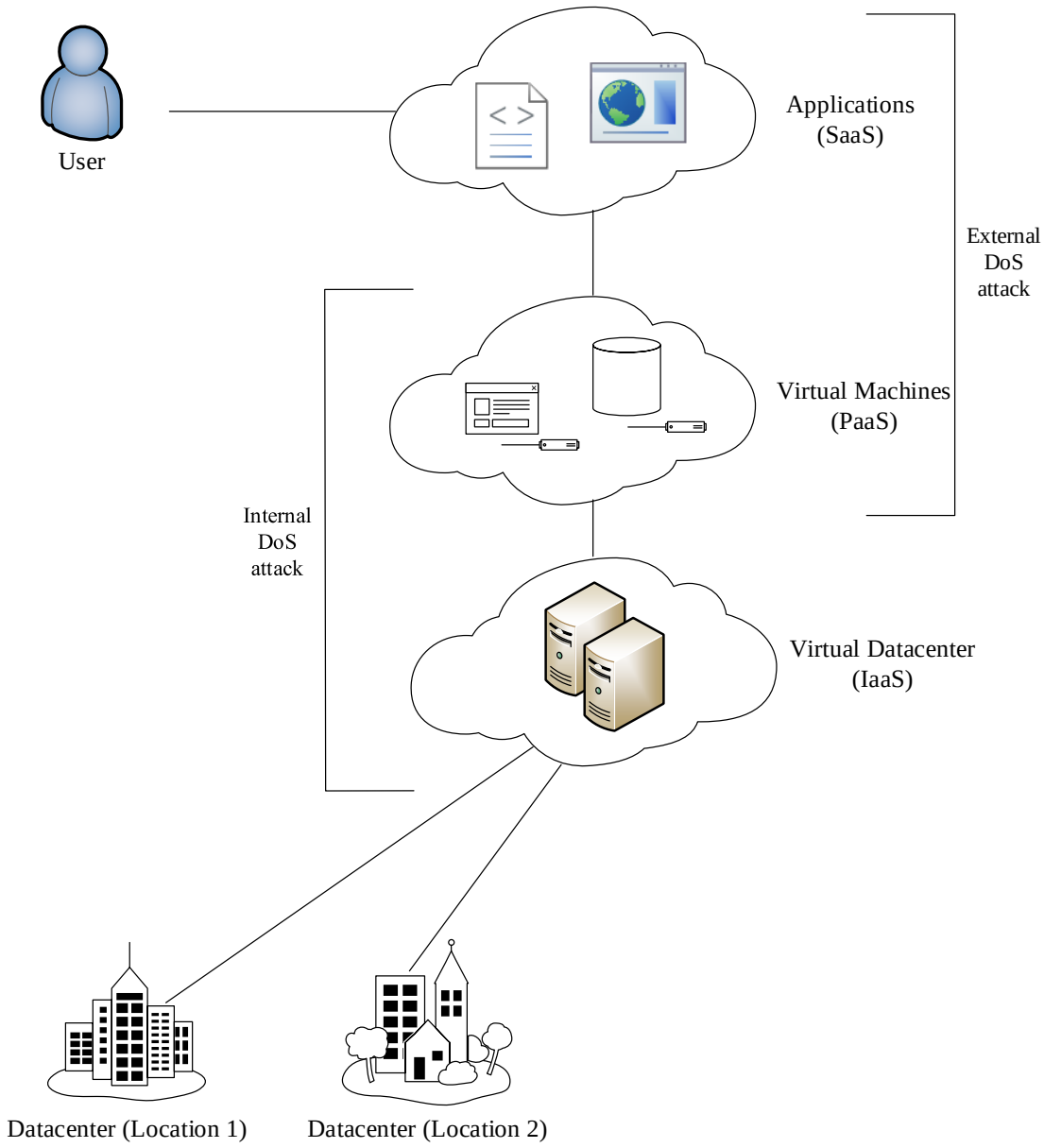

Figure 2.7: External and internal cloud-based DoS attacks

Based on an investigation of the major types of DoS attacks, we derive a taxonomy of cloud-based DoS attacks, as illustrated in Table 2.1. Our classification is focused on cloud-computing aspects, such as a cloud-based type of attack, as well as on recommended practical defense mechanisms and the drawbacks of each mechanism.

Table 2.1: Types of DoS attacks on cloud systems

Attack	Cloud-based Type	Recommended Practical Defense Mechanism	Limitation
IP Spoofing	External Internal	- Hop Count Filtering (HCF) in the PaaS model (Wang, Jin, & Shin, 2007)	- The attacker can build his/her own IP2HC mapping to avoid HCF
	External Internal	- Trust-based approach in the IaaS model (Gonzalez, Anwar, & Joshi, 2011)	- Another compatible solution should be proposed to detect IP spoofing in distribution routers
SYN Flooding	External Internal	- SYN cache approach in the PaaS model (Lemon, 2002)	- Increase in latency
	External Internal	- SYN cookies defense approach in the PaaS model (Lemon, 2002)	- Lowers the performance of the cloud system
	External Internal	- Reduces the time of SYN messages received in the PaaS model	- Some of the legitimate ACK packets could be lost
	External Internal	- Filtering mechanism in the IaaS model	- Not reliable due to the limited use of this method
	External Internal	- Firewall mechanism in the IaaS model	- May affect the performance of the networking system in the cloud
	External Internal	- Active monitoring mechanism in the IaaS model (Schuba et al., 1997)	- Decreases resource performance in the cloud

Smurf Attack	External Internal	- Configuring VMs in the PaaS model	
	External Internal	- Configuring network resources in the IaaS model	
Buffer Overflow	External Internal	- Prevented when writing a source code mechanism in the SaaS model (Fu & Shi, 2012)	- Time consumption
	External Internal	- Performs the array bounds checking mechanism in the SaaS model (Fu & Shi, 2012)	- Decreases resource performance in the cloud
	External Internal	- Runtime instrumentation mechanism in the SaaS model (Fu & Shi, 2012)	- Not reliable
	External Internal	- Analyzes the static and dynamic code mechanism in the SaaS model (Fu & Shi, 2012)	- Not reliable
Ping of Death	External Internal	- May not currently affect any cloud service model; however, the attack could be developed in the future	
Land.c	External Internal	- May not currently affect any cloud service model; however, the attack could be developed in the future	
Teardrop.c	External Internal	- May not currently affect any cloud service model; however, the attack could be developed in the future	

2.3.2 Cloud Computing Authentication and Authorization Protocols

The identity authentication process is considered the principal gateway to cloud-based services. Therefore, these gateways have become increasingly susceptible to adversaries who may use DoS attacks to permanently close these gateways. Numerous authentication protocols exist that can verify identities and protect conventional networked applications. However, these authentication protocols may themselves introduce DoS risks when used in cloud-based applications. This risk introduction could occur on account of the utilization of a heavy verification process that can consume the cloud resources and disable the application service.

This section provides an example of one of these protocols. In addition, some of the proposed authentication protocols utilized in cloud computing are presented. Furthermore, the OAuth 2.0 protocol, which is a widely used authorization protocol for cloud systems, is described. Finally, the Host Identity Protocol (HIP), which is a DoS-resistant authentication protocol in conventional network systems, is explained.

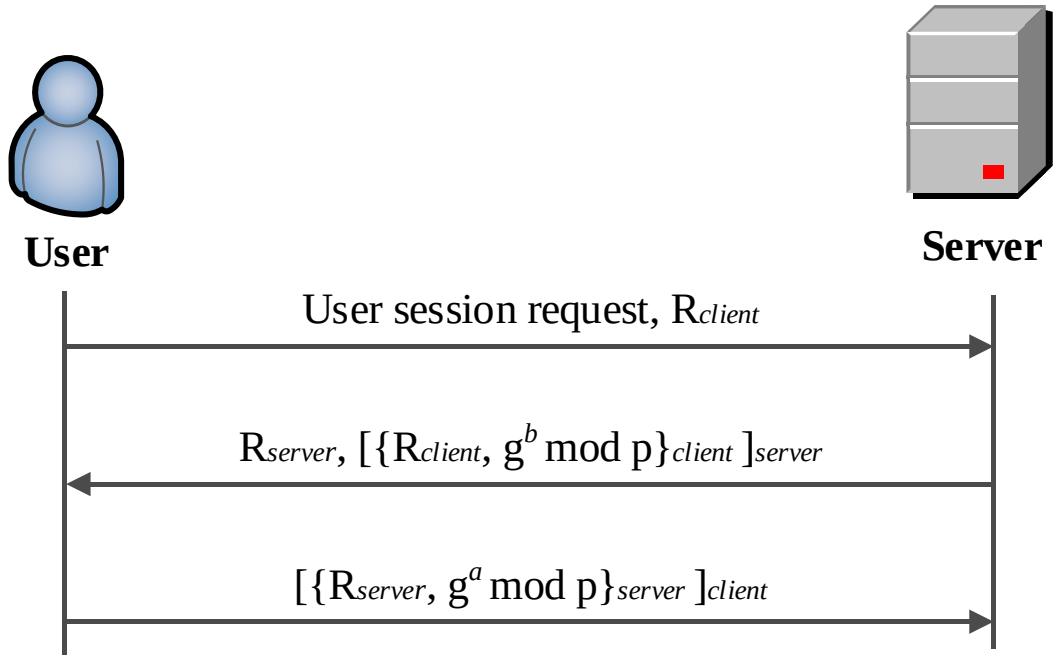

Figure 2.8: Mutual authentication protocol

An example of authentication protocols that can independently introduce internal DoS risks is shown in Fig. 2.8. The goal of this protocol is to cross-authenticate the user and server. This protocol uses the ephemeral Diffie-Hellman key exchange (Diffie & Hellman, 1976), where a, b, p, and g are the values of Diffie-Hellman, as shown in Fig. 2.8. In this protocol, once the server receives a request from a user, the server begins generating the secret value, b. Subsequently, the server computes the exponential value, g^b mod p. Moreover, the server encrypts the nonce of the user and the exponential value via the user public key. Finally, the server digitally signs the encrypted message. All of these processes are executed by the server, which consumes considerable resources without determining whether the request is legitimate. This mutual authentication, which is vulnerable to the DoS attack, is similar to the two-way authentication version of the Transport Layer Security (TLS) protocol (Dierks, 1999).

Another example of a protocol that independently introduces DoS risk is Kim et al.'s protocol (Kim, Fujioka, & Ustaoğlu, 2009), which aims to securely authenticate the key exchange between participants. In this protocol, once the server receives the first message, the server begins computing an exponential value and generates the key. Accordingly, the server resources can become exhausted by the initial requests.

Many authentication protocols have been proposed for the cloud-computing environment. However, they do not protect against DoS attacks. Yassin et al. (A. A. Yassin, Jin, Ibrahim, Qiang, & Zou, 2013) proposed an authentication process that uses a one-time password for mutual authentication of the user and cloud server. Although Yassin et al.'s authentication scheme defends against replay attacks, it cannot defend against DoS attacks. Other cloud-based authentication protocols for DoS prevention have been proposed (e.g., Choudhury et al. (Choudhury, Kumar, Sain, Lim, & Jae-Lee, 2011), Hwang et al. (Hwang, Chong, & Chen, 2010), Jaidhar (Jaidhar C. D, 2012), and Tsaur et al. (Tsaur, Li, & Lee, 2012)). These protocols use a smart card reader for the authentication process. Therefore, the cloud users should use a card reader device for each authentication by the cloud server. Additionally, the scheme of Yassin et al. (A. a. Yassin, Jin, Ibrahim, & Zou, 2012) recommends the use of another third-party device, such as a fingerprint scanner.

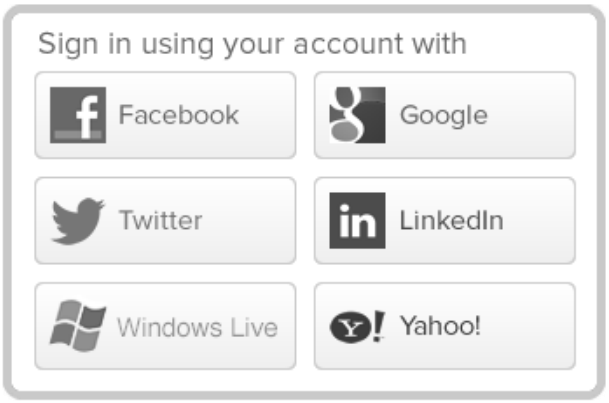

Figure 2.9: Example of social authentication icons (Darwell, 2013)

Furthermore, the OAuth 2.0 authorization framework was developed by Hardt in 2012 (Hardt, 2012) as an improvement on the previous OAuth 1.0 protocol (Hammer-Lahav, 2010). Currently, it is implemented as an authorization framework (as OAuth) by many vendors, such as Facebook, Google, Twitter, LinkedIn, and Yahoo (Boyd, 2012), as shown in Fig. 2.9. This protocol was developed to control third-party client accessibility to HTTP services. For instance, assume that a user would like to play the Angry Birds game and post the score and screenshots on his/her Facebook account. In this scenario, the user must give the Angry Birds app access to his/her Facebook account so the app can obtain the user's information and post the game score on the user's behalf. By employing OAuth, this can be done in such a way that, when the app needs to access any Facebook services (i.e., posting), OAuth will redirect the user back to the Facebook login screen to login into the user's Facebook account. In this case, the user gives his/her username and password to Facebook itself, not to the Angry Birds app. Facebook will then ask whether the user wants to authorize this app. It will then create an access token that works like a password. This token only allows the app to access the user's Facebook account information and post the user's score when needed.

Thus, the OAuth 2.0 protocol permits a third-party client, such as an application, to access a server's resources (user profile information) with rules and permissions in a way that avoids exchanging the user's credentials. The OAuth 2.0 protocol participants are as follows:

- **Client:** An application (e.g., Angry Birds app) that uses an application program interface (API) (Siriwardena, 2014) to access the resource owner's (user's) protected resources (profile) with his/her authorization.

- **Resource owner:** The user of the application who grants access to his/her protected resources (profile) that are available on the resource server (e.g., a Facebook user profile server).

- **Resource server:** The server (Facebook user profile server) that hosts the protected resources of the user (his/her profile). Typically, this server provides the API and hosts, and protects the user's data (profile).

- **Authorization server:** The server that receives the resource owner's (user's) permissions to generate the access token and then sends it to the client (Angry Birds app). Therefore, the client can access the protected resources (user's profile). The authorization server (Facebook server) can be the same as the resource server (Facebook user profile server).

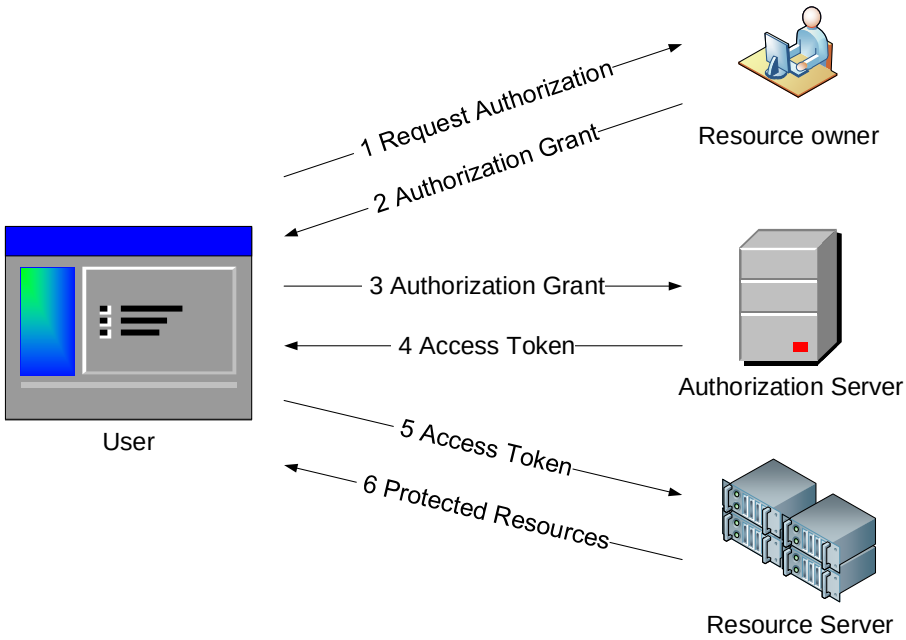

Figure 2.10: Overview of the OAuth 2.0 protocol

As shown in Fig. 2.10, the OAuth 2.0 protocol works as follows:

1. The client (Angry Birds app) sends an authorization request to the resource owner (user).
2. The resource owner (user) sends an authorization grant to the client (Angry Birds app). However, the type of authorization granted is determined based on the authorization request received from the client. Examples of authorization grant types are as follows:
    - **Authorization code grants:** In this type of grant, the resource owner (user) grants an authorization code to the client (Angry Birds app) after an authorization server (Facebook server) authorizes the resource owner. The client uses this authorization code and therefore does not need the credentials of the resource owner. An authorization code grant is a secure type of grant.
    - **Implicit grants:** This type of grant is implemented on the client (Angry Birds app) using software language so that the access token is immediately given to the client. Although the implicit grant reduces the protocol overhead, it may introduce security threats.
    - **Resource owner password credential grants:** For this type of grant, the access token is given to the client (Angry Birds app) using the credentials of the resource owner (username and password). This grant type can be implemented when the resource owner (user) completely trusts the client.
    - **Client credential grants:** This type of grant can be implemented to limit access to the server's secure resources, such as Facebook user profiles. Therefore, it can be implemented when the client (Angry Birds app) is the owner of the secure resource (Facebook user profiles) or when the client has been granted permission to access the secure resource in the past.
3. The client (Angry Birds app) sends the received authorization grant to the authorization server (Facebook server) to obtain an access token.
4. The authorization server (Facebook server) authenticates the client (Angry Birds app) and validates the authorization grant. It then issues an access token that will be known by the resource server (Facebook user profile server) and sends it to the

client. This access token is used instead of the standard username and password authentication. The access token has an expiration time. Once it expires, an optional refresh process can be implemented by reapplying the third step of the protocol.

5. The client (Angry Birds app) sends the received access token to the resource server (Facebook user profile server) so that it may access the secure resource (user's profile data).
6. The resource server (Facebook user profile server) validates the access token and then allows the client (Angry Birds app) to access the resource (user's profile data). For instance, the client could share the game's achievements of the user on his/her Facebook timeline.

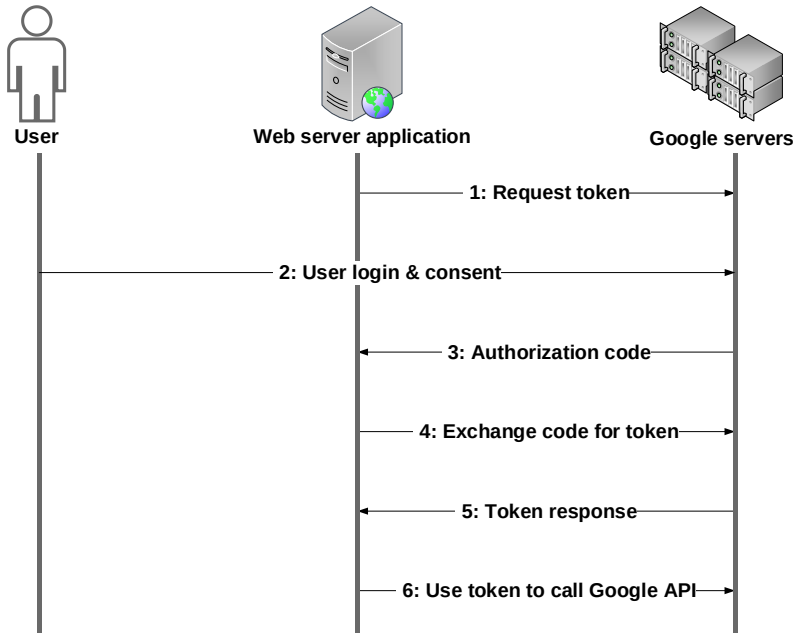

Figure 2.11: Authorization process of Google's OAuth 2.0 for web server applications

However, if the OAuth 2.0 protocol is not securely implemented, the protocol could introduce security breaches into the system, such as in Google's implementation of the OAuth 2.0 protocol (Google Developers, 2014). This implementation of the OAuth 2.0 protocol for web server applications was developed so that Google's API could be implemented in web server applications and frameworks, such as game applications. As

shown in Fig. 2.11, the authorization process of the implemented protocol works as follows:

1. The web server application (app) requests the token.
2. The user is redirected to the uniform resource locator (URL) that is assigned by Google. This URL contains some information, such as access type.
3. Google servers perform the authentication task. Once Google servers authenticate the user, the web server application (app) receives the authorization code.
4. To obtain an access token, the web server application (app) exchanges the user's data and obtains an authorization code with the Google servers. To enable future offline access, the web server application receives the refresh token first; this token information is then exchanged with the Google servers to obtain the access token.
5. The Google servers send the access token to the web server application (app).
6. The web server application (app) uses the received token to call Google's API.

For offline access, in this implementation, the refresh token is stored on the web application server's storage device for future communication. The stored refresh token remains valid until it is revoked by the user (Google Developers, 2014). These stored data can lead to security flaws that exhaust the storage resources of the web servers, as detailed in Section 5.2.2.

The Host Identity Protocol (HIP) (Moskowitz, Nikander, Jokela, & Henderson, 2008) is an example of an authentication protocol that is used to identify a DoS attack on a conventional network. However, this protocol cannot be implemented in the application layer to defend against external DoS attacks. This is because HIP is based on the host identity on the network layers in the Open Systems Interconnection (OSI) reference model. Moreover, it is configured and controlled at an operating system level. Furthermore, any authentication protocol that is based on IP address verification, such as the Internet Protocol Security (IPSec) protocol, makes it difficult to hide the identity of the participants. HIP solves the DoS-SYN flooding attack that was presented in Section 2.2 by creating additional processing to establish a new TCP connection between two participants. This new four-way handshake process is based on a crypto-puzzle that requires the user to

reverse a hash function. Consequently, the puzzle-solving process forces the user to complete some computational operations. Verifying the puzzle solution is a short operation on the server side.

2.3.3 Authentication Protocol Validation

Burrows, Abadi, and Needham (BAN) proposed a belief logic (Burrows, Abadi, & Needham, 1989) to analyze the security requirements of authentication protocols. Subsequently, logic by Syverson and Van Oorschot (SVO) (P. F. Syverson & Van Oorschot, 1994) was introduced as an extension model to address some of the limitations of BAN. SVO uses some of its own notations in addition to those used in BAN. The analysis steps of any authentication protocol using SVO (P. Syverson & Cervesato, 2001) are as follows:

1) Initial assumption: An assumption of the initial status of the protocol.
2) Received message assumption: An assumption regarding the messages each party receives when the protocol is completed in a trusted fashion.
3) Comprehension assumption: An assumption regarding what the receiver believes, and what parts of the received message are unknown.
4) Interpretation assumption: An assumption of how each party interprets the received messages.
5) Derivation: It derives the analysis goals using the previous assumptions.

For further security validation, any protocol design in general should be investigated in terms of its vulnerability to DoS attacks using the cost-based model approach proposed by Meadows (Meadows, 2001). This approach aims to prevent DoS attacks during the authentication process. It depends on the exhausted resource costs of the participants. The cost-based model approach logically demonstrates the effectiveness of the protocols in preventing DoS attacks. The computation cost is defined as the total resource usage cost of the requester (user) and responder (server) when both participate in the authentication protocol. The cost is computed during the process until a DoS attacker is detected and is prevented from participating. The total cost of the requester is the total estimated cost of each operation involved in the authentication process on the requester’s side until the

authentication process ends. However, the total cost of the responder is the total estimated cost of each operation during the authentication process until the requester is determined to be either a legitimate requester or attacker.

Meadows (Meadows, 2001) proposed the categories of inexpensive, medium, and expensive for an operation's cost. This approach assumes that the exponential, signature check, and signature operations performed during an authentication process are expensive. The pre-calculated exponential value, and the encryption and decryption operations are medium cost. Any other operations are inexpensive.

2.3.4 Review Summary

The literature review on DoS-aware cloud-based authentication protocols is summarized in Table 2.2.

Table 2.2: Literature review summary

Proposed Protocol	DoS awareness	Implementation environment	Uses third-party device
OAuth 2.0 (Hardt, 2012)	Based on the implementation	Conventional network and cloud computing	No
HIP (Moskowitz et al., 2008)	Yes	Conventional network	No
Yassin et al. (A. A. Yassin, Jin, Ibrahim, Qiang, & Zou, 2013)	No	Cloud computing	No
Choudhury et al. (Choudhury et al., 2011)	Yes	Cloud computing	Yes (smartcard reader)
Hwang et al. (Hwang et al., 2010)	Yes	Cloud computing	Yes (smartcard reader)
Jaidhar (Jaidhar C. D, 2012)	Yes	Cloud computing	Yes (smartcard reader)
Tsaur et al. (Tsaur, Li, & Lee, 2012)	Yes	Cloud computing	Yes (smartcard reader)
Yassin et al. (A. a. Yassin, Jin, Ibrahim, & Zou, 2012)	Yes	Cloud computing	Yes (fingerprint scanner)

Based on the implications of the above literature review summary, an authentication protocol suite against DoS attacks is herein proposed that works in the cloud-computing environment. Moreover, it considers the security requirements of the authentication protocol without using third-party devices.

Chapter 3

3 Cloud-Based Secure Authentication Module 1 (CSAM-1) Protocol for Private and Community Cloud Computing

In this chapter, the first module of the proposed cloud-based secure authentication (CSAM-1) protocol suite to defend against external DoS attacks is presented. The CSAM-1 protocol is developed so that the total computational cost of the user side is greater than the resource operations cost of the cloud-based server when they jointly participate in the authentication process. The main objective of the CSAM-1 protocol is to defend against external DoS attacks. Therefore, CSA can be implemented and configured in private and community cloud computing because the infrastructure and platforms in these deployment models are secured by the organizations. On the other hand, in public and hybrid cloud computing models, different infrastructures are communicated together. Thus, there is no guarantee that all infrastructures have the same security level. The solution to this problem is addressed in Chapter 4. Table 3.1 shows the notations that are used in the CSAM-1 protocol suite.

Table 3.1: Notations of the CSAM-1 protocol

Notation	Description
Cloud_user	Cloud user/client
Cloud_server	Cloud server/service provider
Cloud_user ID (CUID)	**Cloud_user** ID
Unique encrypted text (UET)	Unique encrypted text; the key of the UET is known only by **cloud_server**
Session key (SK)	Session key
A	A set of random integers of the server challenge function
S	A subset sum of the server challenge functions
B	A binary vector representing the challenge function solution
$R_{\mathbf{cloud\_server}}$	The nonce that is generated by **cloud_server**
T	Timestamp
K_X	Secret key of *X*
MK	Master secret key of **cloud_server**
Tag	An encryption of timestamp and cloud user ID by master secret key of **cloud_server**

CSAM-1 consists of two protocols, as shown in Fig. 3.1. The first protocol is used for the registration process, which is an agreement process between the participants (**cloud_user** and **cloud_server**) about specific shared information. Thus, the participants can use that information during the operation of the second CSAM-1 protocol. The second protocol is an adaptive-based identification and authentication protocol that works against DoS attacks. This protocol is developed based on the cost-based model approach. In addition, this protocol is used for the authentication process, which includes all operations that occur on the basis of initially agreed upon information of the previous protocol. As a result,

cloud_server can confirm the identity of **cloud_user** and then completes the authentication process, or it can detect and then prevent an intruder in the case of a DoS attack.

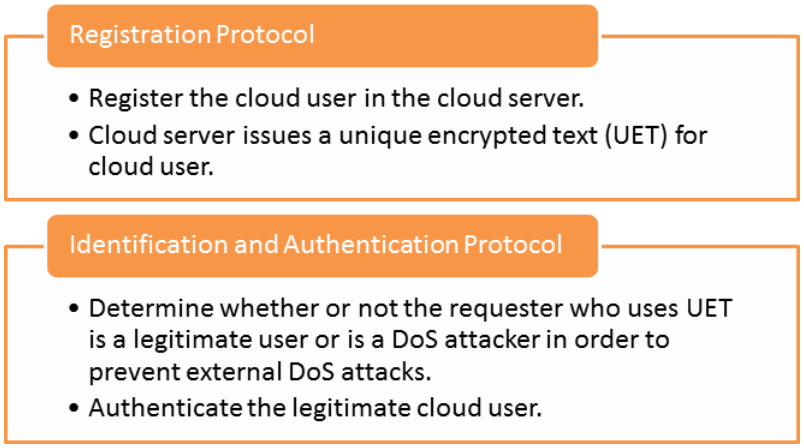

Figure 3.1: CSAM-1 protocols

3.1 Registration Protocol

In the CSA registration protocol, **cloud_user** and **cloud_server** share the required identity data to register **cloud_user** in the **cloud_server** database.

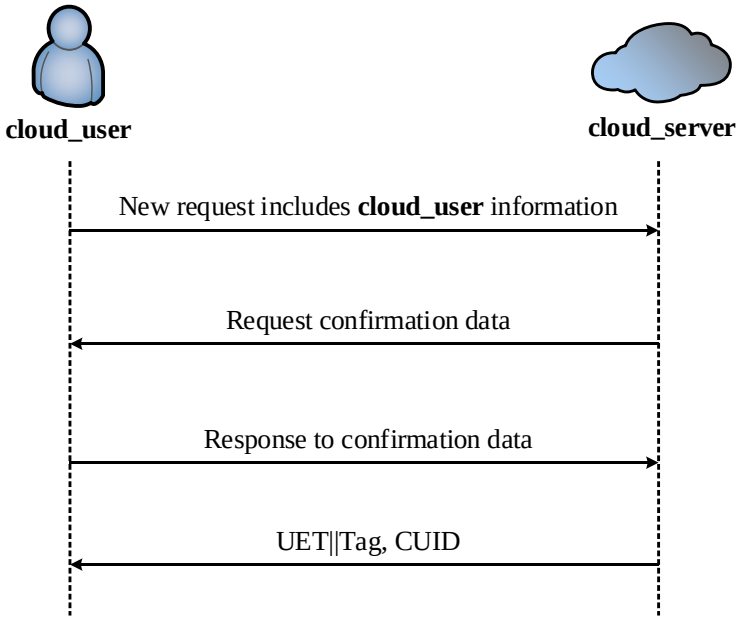

Figure 3.2: Registration protocol in CSAM-1

As shown in Fig. 3.2, the registration process begins when **cloud_user** submits all the required information to **cloud_server**. This information includes the first name, last name, organization name, email address, and any other information that is required by the cloud service provider. **Cloud_server** validates the received information, stores it in a database, and then sends a validation email message to **cloud_user** to confirm **cloud_user**'s information. If **cloud_user** does not confirm his/her information after a specific amount of time, **cloud_server** deletes **cloud_user's** information. After validation, **cloud_server** activates **cloud_user**'s account. At the same time, **cloud_server** generates a CUID and a UET. This UET is encrypted by **cloud_server**'s master key (MK), which is known only by **cloud_server**.

Figure 3.3: UET structure in CSAM-1

As shown in Fig. 3.3, the UET contains **cloud_user** information and other information that is created by **cloud_server** during the processes of the CSA protocols, such as the CUID, last session key, and last timestamp. The UET is a piece of information that is not stored on **cloud_server**; rather, it is sent to the requesting **cloud_user**.

CUID	**Tag** = E (T + CUID, MK)	**PSK**	Other **cloud_user**'s information

Figure 3.4: Cloud_user (lookup) database table structure

To identify **cloud_user** and to avoid the risk of DoS attacks**, cloud_server** generates a tag that contains a timestamp (T) and CUID that are both encrypted by **cloud_server**'s MK. C**loud_server** then adds CUID and tag information to **cloud_user**'s table in its database (referred to as the lookup table), as shown in Fig. 3.4. It should be noted that the number of entities in the lookup table depends on the number of confirmed registered **cloud_users**.

In the next step, **cloud_server** sends UET along with the tag to **cloud_user**. Once **cloud_user** receives the required data from **cloud_server**, both **cloud_user** and **cloud_server** agree upon the pre-shared key. The pre-shared key is created using a key derivation function and a shared secret. **Cloud_user** and **cloud_server** agree upon the key derivation function and a shared secret at the end of the registration protocol, which is exchanged via a secure channel in a very restricted environment. This approach is the same approach of the pre-shared key (PSK) agreement that used in the UMTS and WPA2 protocols (Southern, Ouda, & Shami, 2013). Consequently, **cloud_user** stores the UET and a pre-shared key for a future authentication process. **Cloud_server** stores PSK in the lookup table.

Even if **cloud_user** is registered to **cloud_server**, **cloud_user** cannot access the services available through **cloud_server** unless **cloud_server** identifies and authenticates **cloud_user**. To perform the identification and authentication protocol that is ready to defend against external DoS attacks, CSAM-1 provides an outer shield to the authentication protocol. This helps in distinguishing legitimate **cloud_users** from DoS attackers. The CSA-adaptive based identification and authentication protocol is designed to provide this outer shield as described in the following section.

3.2 Identification and Authentication Protocol

The adaptive-based identification and authentication protocol utilizes the cost-based model approach. Before applying the computational power of the authentication protocols in the server side, users are asked to prove their sincere commitment for receiving the **cloud_server** services. This validation of commitment can be achieved by any technique that can force the users to utilize a significant amount of computational power before the servers utilize them in order to confirm their genuine requests. Currently, "client puzzles" is a common technique that realizes the cost-based model approach (Juels & Brainard, 1999).

In this research study, a technique based on a one-way function is proposed to realize the cost-based model approach. A cryptographic knapsack problem has been chosen because it is a strong one-way function, while also being flexible for adaptability (Salomaa, 1996).

The knapsack problem is strong because it is known to be an NP-complete problem (Manber, 1989). It is to be noted that the problem is considered as NP-complete when the problem is in both NP (nondeterministic polynomial time) and NP-hard. In addition, the main characteristic of the selected one-way function in this protocol is an adjustable difficulty level to solve the puzzle based on the efforts of the users.

In cryptography, a knapsack problem is described as follows. Given a set of positive integers (i.e., items) $A = a_1 \ldots a_n$ and a positive integer value S. It needs to be determined if there is non-empty subset of $a_1 \ldots a_n$ whose values sum to S. For example, let the set of items in knapsack A be (13, 54, 28, 73, 3, 36) and the summation S be 89. Therefore, finding the elements 13, 73, and 3 solves the problem because their summation is equal to 89. In other words, finding a binary vector, B, such that A . B = S solves the problem. In this example, B is the vector (1, 0, 0, 1, 1, 0); therefore, A . B = 13 + 73 + 3, which is 89.

Typically, the complexity of the knapsack problem depends on the size of knapsack A (the number of its items; for example, n) and on the number of ones (for example, m) in binary vector B. If the number of items in A (i.e., n) is small, then an exhaustive search for the solution is practical. Furthermore, if the number of ones in B (i.e., m) is small compared to n, then a solution can be found in a reasonable time via dynamic programming algorithms. Consequently, by adjusting the values of n and m, determining the difficulty level of the knapsack problem, and hence the cost-based approach, can be adaptively realized.

The CSAM-1 protocol considers n = 512 items in the knapsack puzzle problem. These items are fixed integer values that both parties should agree upon during the registration protocol. Based on the experimental result (see Section 5.2.1), obtaining vector B forces **cloud_user** to become involved in finding the solution to 2^{512} subsets, which is a very large time interval and significant resource consumption. The number of subsets of items is adjustable based on the required efforts of the participants. Moreover, the chosen items that are used during the summation process are determined by hashing the values of CUID, MK, and $R_{\mathbf{cloud\_server}}$ using SHA2-512, where MK is the master secret key of **cloud_server**. The result of the hash function is a 512-bit stream. Moreover, the subset of the 512-bit stream that includes a specific number of ones (m) represents the required vector, B, of the

knapsack problem. For example, if the protocol is developed to let m = 55, **cloud_server** takes the subset of the 512-bit stream that includes the first 55 ones. Increasing the value of m makes the process of solving the puzzle more difficult, which would magnify the time-consuming nature of the puzzle-solving process. The hashing process is mandatory to verify the subset summation value (S) of **cloud_user** after the calculation process.

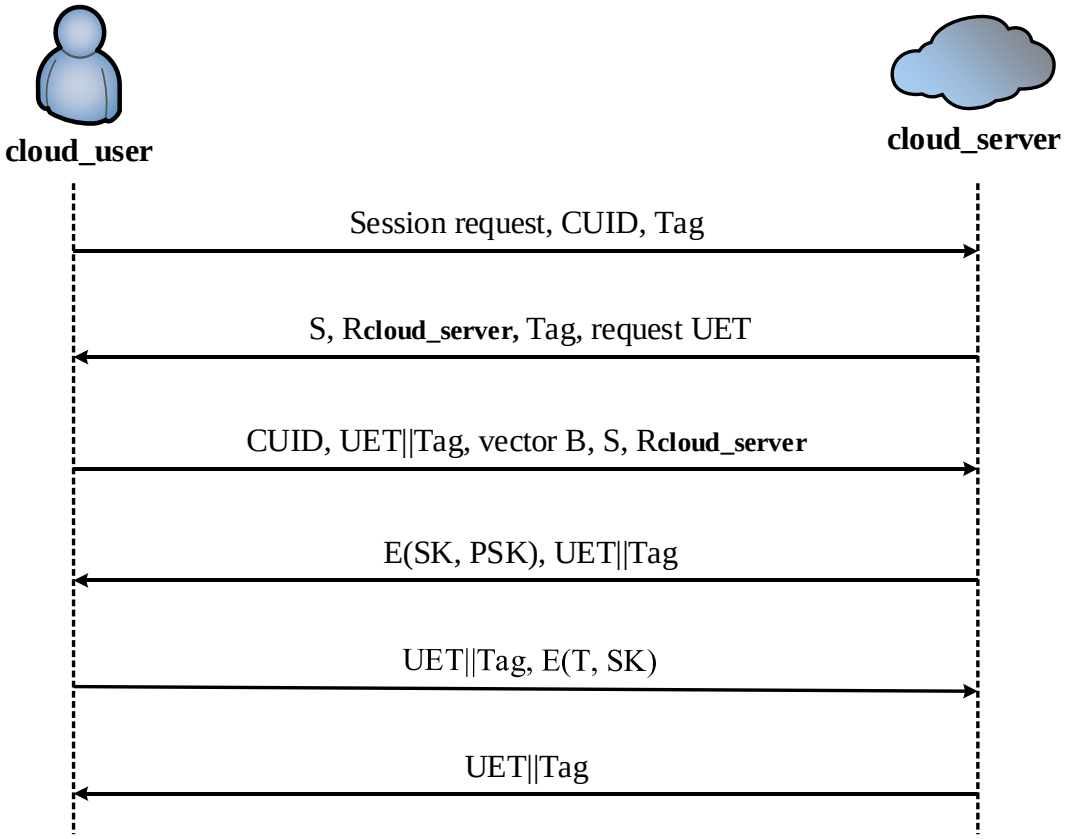

Figure 3.5: Adaptive-based identification and authentication protocol in CSAM-1

The function of this adaptive-based identification and authentication protocol process shown in Fig. 3.5 is described as follows:

Cloud_user sends a request for a service along with the CUID and tag to **cloud_server**. At this point, **cloud_server** blocks any CUID that has performed three consecutive requests within a low time threshold to prevent DoS attacks. The attacker may attempt to launch a DoS attack by sending requests with randomly generated CUID values. In this case, the

cloud_server easily check the received tag with the stored tag in the lookup table. If these are not identical, the request is considered an attacker request. It should be noted that **cloud_server** is not required to perform an encryption process to check the tag value.

Cloud_server directly replies to **cloud_user** by sending the puzzle element as a challenge, which is the subset summation value (S) along with a **cloud_server** nonce ($R_{\mathbf{cloud\_server}}$) and a new generated tag. **Cloud_server** generates a new tag value and updates the lookup table in the database each time it checks the tag value to preserve the refreshment property of the protocol.

Cloud_server asks **cloud_user** to prove its sincere commitment to receiving the **cloud_server** services by asking for the UET as well as the puzzle solution to the (S) value. The expected solution for this challenge is vector B.

Once **cloud_user** calculates and obtains vector B, it sends the CUID and UET, received tag, vector B, S value, and received $R_{\mathbf{cloud\_server}}$ to **cloud_server** for validation. At this point, **cloud_server** has all the information required to validate the authentication requests; therefore, **cloud_server** can apply the validation process to only a few operations, such as those outlined below.

First, **cloud_server** validates the received tag by comparing it with the stored tag in the lookup table. Second, **cloud_server** checks the subset of item a_i by securely hashing (CUID, MK, $R_{\mathbf{cloud\_server}}$) and comparing the resulting vector with received vector B to determine whether they are identical.

If any of the two previous conditions do not apply, **cloud_server** drops the request and considers it to be an attacker's request. However, once the **cloud_user** request passes both conditions, **cloud_server** decrypts the UET and validates the decrypted information containing the CUID. Then, **cloud_server** generates a new tag and updates its field in the lookup table.

To complete the authentication process once the protocol determines that **cloud_user** is legitimate, both participants agree on the SK for future interactions. In addition, they can agree on the sub-session key if they later require a refreshment process. Therefore, **cloud_server** generates the SK, which is encrypted via a pre-shared key. Moreover, **cloud_server** adds both the SK and T information to the UET. Consequently, **cloud_server** is protected against DoS attacks on the storage space because the UET is never saved in the **cloud_server**. Furthermore, **cloud_server** can apply the refreshment property of the session key for future communication by adding the SK to the UET.

Therefore, **cloud_server** sends the generated SK to the **cloud_user** that is encrypted by the pre-shared key, along with the modified UET and newly generated tag. It should be noted that the notation E(SK, PSK) means that the SK is encrypted by the PSK.

Cloud_user first confirms the received encrypted SK by encrypting timestamp T using SK. It then returns encrypted SK, along with the received UET and tag, to **cloud_server**. Therefore, **cloud_server** decrypts the UET, validates the CUID, and obtains the SK. It then confirms SK by decrypting the received timestamp T using the SK.

Later, the two parties can agree regarding the sub-session keys by re-applying the processes of the authentication protocol. Accordingly, **cloud_server** can generate a sub-session key and add it to the UET without storing it in the cloud system.

3.3 Discussion

The authentication protocol suite is proposed to identify and authenticate cloud users in the SaaS model and provide a strong shield against external DoS attacks. By integrating the client puzzle problem and utilization of the UET, security breaches that may lead to DoS attacks can be avoided.

In the CSAM-1 protocol suite, we rely on the computational complexity theory to determine different levels of client-puzzle solution difficulties. Thus, the identity protocol is designed to minimize the computational cost incurred by the cloud resources; moreover, the computation cost incurred by cloud users is adjustable based on the service's sensitivity. The high computational cost influences an attacker launching a DoS attack with

a massive number of requests from his/her device. However, if the attacker uses many different devices to launch DoS attacks, the cloud system will not be exhausted because the attack can be detected at an early stage of the authentication process.

In practical terms, the proposed CSAM-1 protocol suite can be implemented in the SaaS model of cloud computing systems because the protocol simply relies on basic hardware and software requirements of both the cloud systems and users.

However, the proposed CSAM-1 protocol suite does not consider the possible DoS risks faced by the other cloud service models, such as PaaS or IaaS. Although the CSAM-1 protocol suite can be implemented on private and community cloud models at this stage, it must be redeveloped for implementation on public or hybrid cloud models, as discussed in Chapter 4.

Chapter 4

4 Cloud-Based Secure Authentication Module 2 (CSAM-2) Protocol for Public and Hybrid Cloud Computing

In this chapter, a second module of the cloud-based secure authentication (CSAM-2) protocol suite is proposed to prevent DoS attacks in PaaS and IaaS cloud-computing service models. In addition, it securely authenticates and identifies cloud users who would like to use cloud services. The CSAM-2 protocol suite relies on four protocols, as shown in Fig. 4.1. These protocols sequentially progress so that each protocol process starts its execution based on the output of the previous protocol. In the registration protocol, the cloud user begins the registration process in the cloud server. Once the user is registered, the cloud server in the identification and authentication protocol determines whether the cloud user is a legitimate user. Once the external DoS attack is prevented, the service management and allocation protocol directs the cloud user to the requested service, which prevents the risks of internal DoS attacks. Finally, once the legitimate user is directed to the host of the requested service, the host session and authentication protocol validates the cloud user to access the requested service.

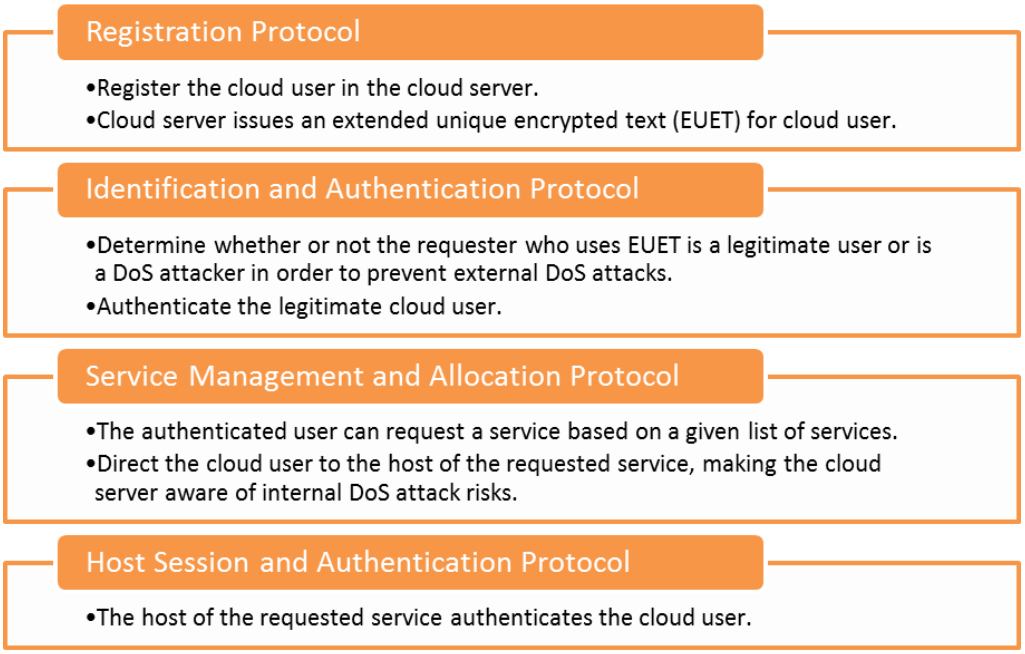

Figure 4.1: CSAM-2 protocols

The notations of the CSAM-2 protocol suite are shown in Table 4.1.

Table 4.1: Notations of the CSAM-2 protocol

Notation	Description
cloud_user	Cloud user/client
cloud_server	Cloud server/service provider
v_service_host	Virtual host of the requested service
CUID	Cloud user ID
SVID	Service ID
ACL	Access control list; an information set issued by **cloud_server** to allow **cloud_user** access to the requested services
EUET	Extended unique encrypted text, the key of which is known only by **cloud_server**
SK	Session key
A	A set of random integers of the server challenge function
S	A subset sum of the server challenge function
B	A binary vector representing the challenge function solution
R_{cloud_server}	The nonce that is generated by **cloud_server**
T	Timestamp
K_X	Secret key of *X*
MK	Master secret key of **cloud_server**
Tag	An encryption of the timestamp and CUID by the master secret key of **cloud_server**
$Tag_{v\_service\_host}$	An encryption of time stamp and CUID by the secret key of **v_service_host**

4.1 Registration Protocol

In this protocol, the cloud server registers and activates the cloud user in its own database. It is to be noted that depends on successful completion of module 1, this protocol is similar to the registration protocol in CSAM-1 with UET structure that is extended to hold extra security information. In this protocol, the cloud server issues an extended unique encrypted text (EUET) for the cloud user to employ in the future protocols' processes. Therefore, a user registration protocol is proposed to enable both participants (**cloud_user** and **cloud_server**) to communicate to share mandatory identification information. Accordingly, it can register **cloud_user** in the **cloud_server** database.

In the user registration protocol, **cloud_user** initiates a request to **cloud_server** that includes the **cloud_user** information, as shown in Fig. 4.2. This information contains, but is not limited to, the **cloud_user** name, phone number, email address, and other information that **cloud_server** should maintain in its database. **Cloud_server** stores the information in its database and then sends an email message to **cloud_user** to validate the email address. Once **cloud_user** responds to the validation message, it is deemed an activated user. On the other hand, if **cloud_user** does not respond to the message within a specified period, the **cloud_user** information is deleted from **cloud_server**. Hence, the goal of registering and activating **cloud_user** is achieved. Once the registration succeeds, **cloud_server** issues a CUID and a EUET that is encrypted by its own MK.

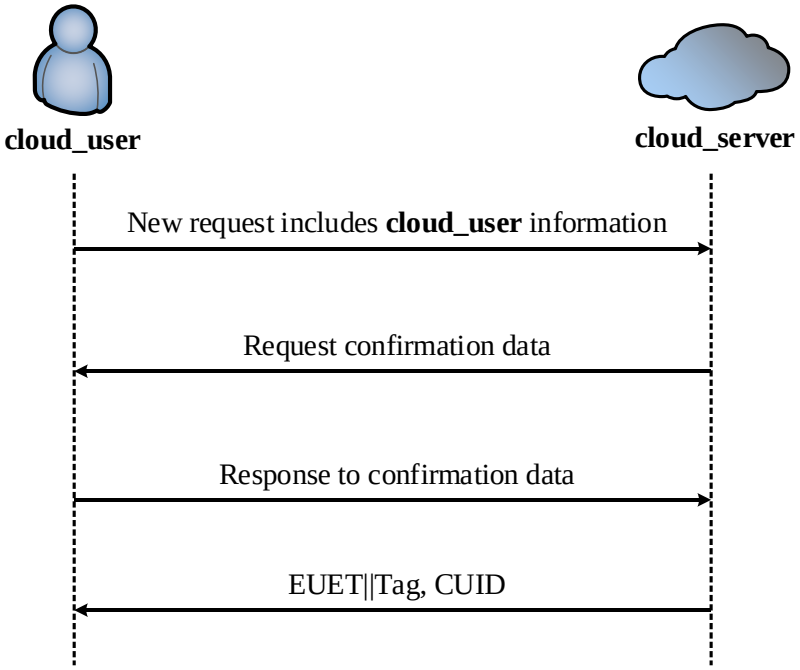

Figure 4.2: User registration protocol in CSAM-2

In addition, **Cloud_server** generates a tag by encrypting the timestamp (T) and CUID by its own MK. C**loud_server** then inserts the CUID and tag into the **cloud_user** information table (lookup table). As mentioned in CSAM-1 (Chapter 3), the number of entities in the lookup table depends on the number of confirmed registered **cloud_users**. In addition, **cloud_server** uses the tag to identify **cloud_user** based on the information in the lookup table and to avoid any risks of DoS attacks.

CUID	Last SK	Last T	Service request status

(a)

CUID	Last SK	Last T	Service request status	Direct accessible SVIDs list	Indirect accessible SVIDs list

(b)

Figure 4.3: (a) EUET structure in CSAM-2 with no service assigned to cloud_user; (b) EUET structure in CSAM-2 with at least one service assigned to cloud_user

Then, **cloud_server** sends the CUID to **cloud_user** along with the EUET and tag. This EUET includes the CUID and other information required for other CSAM-2 protocols, as shown in Fig. 4.3. The EUET is sent to **cloud_user** but is not saved on **cloud_server**.

Both participants use a shared secret and a key derivation function in a restricted and secured environment based on a pre-shared key (PSK) agreement. This agreement process is similar to those used in WPA2 and UMTS protocols (Southern et al., 2013). **Cloud_server** then stores the PSK in the lookup table.

As a result, **cloud_user** in the registration protocol is registered and activated in **cloud_server**. Furthermore, **cloud_server** sends a EUET to **cloud_user**. In the following protocol, **cloud_server** identifies and authenticates the requester who uses the EUET. It is simultaneously aware of external cloud-based DoS attacks.

4.2 Identification and Authentication Protocol

This section presents the proposed adaptive-based identification and authentication protocol. Once again, based on successful completion of module 1, this protocol is similar to the identification and authentication protocol in CSAM-1 with extended UET structure to hold extra security information. In this protocol, **cloud_server** determines whether the requester is a legitimate **cloud_user**. It then authenticates the legitimate **cloud_user**. In this protocol, **cloud_server** forces **cloud_user** to perform a computational process before **cloud_server** is involved in any computational power.

As presented in Fig. 4.4, the steps of the adaptive-based identification and authentication protocol are as follows:

Cloud_user sends an initial session request with the CUID and tag to **cloud_server**.

Cloud_server prevents requests from the same CUID once the consecutive failures reach the maximum allowed limit (three) in a short timeframe. **Cloud_server** identifies the CUID by searching the lookup table in the database and then comparing the received tag to the stored tag value in the table. If they are not identical, the request is considered an illegitimate request; else the CUID is identified by **cloud_server**. It is to be noted that

cloud_server is not required to decrypt the tag until the tag value is validated in the lookup table.

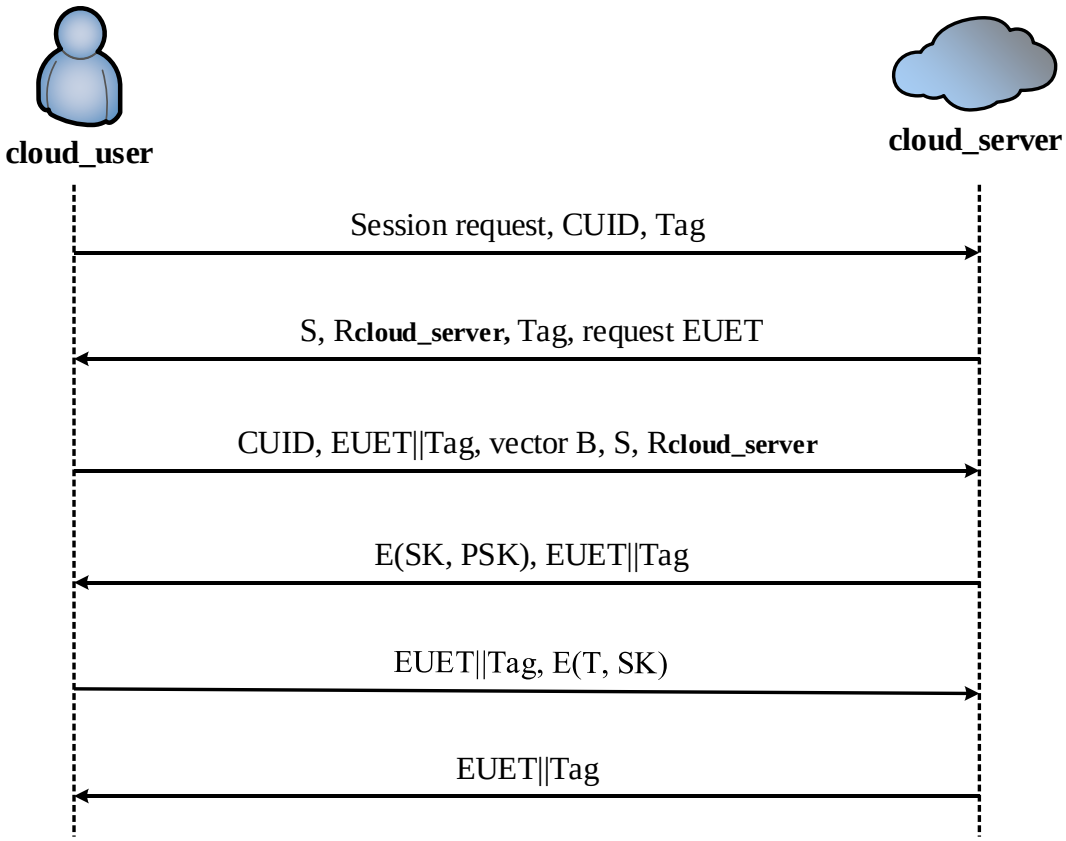

Figure 4.4: Adaptive-based identification and authentication protocol in CSAM-2

Cloud_server responds to the request by sending a puzzle challenge of value S, sending a nonce (R_{cloud_server}), and requesting the EUET from **cloud_user**. In addition, **cloud_server** generates a new tag value and updates the lookup table in the database. The new tag generation process is repeated each time **cloud_server** checks for the tag value so that the refreshment property is achieved in this protocol. **Cloud_server** then sends the newly generated tag to **cloud_user**. To prove the commitment of **cloud_user**, **cloud_server** requests the expected puzzle solution (vector B) along with the EUET and tag from **cloud_user**.

Cloud_user computes the puzzle solution (vector B) and sends it along with the S value, received nonce (R_{cloud_server}), CUID, and EUET in conjunction with a tag to **cloud_server**.

Cloud_server validates the received tag and CUID by checking the lookup table. **Cloud_server** then verifies whether vector B is identical to the result obtained by secure hashing (CUID, $R_{\textbf{cloud\_server}}$, MK). The MK size should not be short to avoid any possibility of using a brute-force attack to guess the MK. Once vector B is verified, **cloud_server** achieves the goal of identifying the legitimate **cloud_user**. On the other hand, if vector B is not verified, the request is rejected and is assumed to be a forgery. **Cloud_server** then decrypts the EUET and checks the CUID registered in the EUET. Furthermore, **cloud_server** issues a new tag, as mentioned in the second step of this protocol.

After **cloud_server** authenticates **cloud_user**, both the participants agree to the session SK for future communications; therefore, **cloud_server** creates the SK and encrypts it using the stored PSK. It then adds it and the current T to the EUET, and it sends this modified EUET in conjunction with the new tag and encrypted SK to **cloud_user**.

To confirm receipt, **cloud_user** encrypts the current T using the SK and returns it and the received EUET along with the received tag to **cloud_server**.

Cloud_server validates the received tag and EUET. It then decrypts the EUET to obtain the registered SK, which is then used to decrypt T. To prevent a DoS attack on storage space, the EUET is not stored in **cloud_server**. In addition, in this case, T is used instead of the nonce because **cloud_server** can verify the operation time of **cloud_user**. Furthermore, c**loud_server** can handle SK's refreshment property for future communications by simply adding the new SK information to the EUET and by re-applying the last three steps of this protocol.

Thus, **cloud_server** in the identification and authentication protocol can prevent external DoS attacks. Additionally, it can identify and authenticate the legitimate **cloud_user**. Therefore, **cloud_user** in the next protocol can request available services from the cloud service provider.

4.3 Service Management and Allocation Protocol

The main goals of the service management and allocation protocol are to enable **cloud_server** to allocate to **cloud_user** the requested available service(s), and to prevent

risks of internal cloud-based DoS attacks. To achieve these goals, a service allocation protocol is herein proposed. In this protocol, **cloud_server** can organize multiple requests for a specific service so that deadlocks cannot occur, which thus protects the services from internal cloud-based DoS attacks.

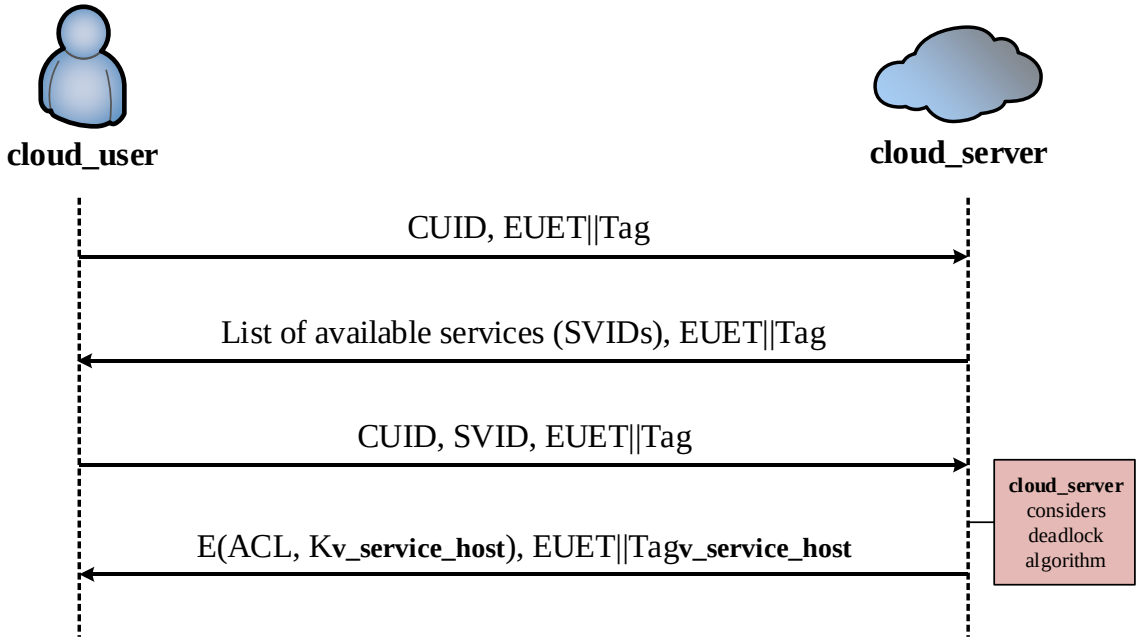

Figure 4.5: Service allocation protocol in CSAM-2

As depicted in Fig. 4.5, **cloud_user** sends the CUID and EUET along with the tag to **cloud_server** and requests available services (SVIDs). **Cloud_server** validates the received CUID by first comparing the received tag with the tag value registered in the lookup table. Upon successful validation, **cloud_server** decrypts the EUET, validates the CUID, and adds the "service requested" status to the EUET. **Cloud_server** then checks for SVIDs that are available in the system and sends a list of available SVIDs to **cloud_user** along with the EUET and a newly generated tag. **Cloud_user** then sends a request for a selected SVID from the received list along with the CUID, EUET, and tag. **Cloud_server** denies any request that includes an invalid tag by comparing the received tag with the one in the lookup table. If the CUID is validated, the requested service might lead to a deadlock problem because many users have multiple accesses to the requested service or other

services through the requested service. Thus, to avoid a deadlock problem and to prevent an internal DoS attack due to a huge number of forged requests for a specific service, **cloud_server** controls the service allocation process by applying a deadlock avoidance strategy. If a request leads to a deadlock problem—for example, by flooding services with false requests—the request is denied. Hence, **cloud_server** prevents internal cloud-based DoS attacks.

(a) *algorithm* **service_request**
input:
integer n: number of services
integer m: number of users
vector available[m]: available[j] = k means there are ***k*** *instances of services type* S_j *available*
matrix max[n,m]: max[i,j] = k means user U_i *may request at most* ***k*** *instances of services type* S_j
matrix allocation[n,m]: allocation[i,j]= k means user U_i *is currently allocated* ***k*** *instances of* S_j
matrix need[n,m]: need[i,j]= k means user U_i *may need* ***k*** *more instances of* S_j *to complete its task*
vector request[]: vector for user U_i, $request_i[j]$ *= k means user* U_i *wants* ***k*** *instances of service type* S_j
output:
Determine the availability of the requested service

```
for i = 0 ... i<n
    for j = 0 ... j<m
        need[i,j] = max[i,j] – allocation[i,j]
if request_i [j] > need_i then
    error: user has exceeded maximum claim
else
    if request_i > available, then
        user must wait, services are not available
    else
        available = available – request_i
        allocation_i = allocation_i + request_i
        need_i = need_i – request_i
        if safe then
            the services are allocated to user
        if unsafe then
            user must wait, and the old service allocation state is restored
```

(b) *algorithm* **safety**
input:
vector work[m]
vector finish[n]
output:
Determine the safety of the system

```
work = available
for i = 0 ... n
    finish[i] = false
for i = 0 ... n
    while finish[i] = false AND need_i ≤ work then
        work = work + allocation_i
        finish[i] = true
for i = 0 ... n
    if finish[i] = true then
        system is safe
    else
        system is unsafe
```

Figure 4.6: (a) Service request algorithm for the user; (b) safety algorithm

One such approach is the banker's algorithm (Dijkstra, 2002), which can be implemented so that the "requested services" replace the "current processes." Moreover, the "required resources" are the directly and indirectly accessible services that **cloud_user** requires, as shown in the algorithm in Fig. 4.6.

CUID
SVIDs SVIDs that are directly accessible with permission
SVIDs Related SVIDs that are indirectly accessible with permission

Figure 4.7: ACL structure

If the service can be allocated to **cloud_user** based on the result of the deadlock avoidance process, **cloud_server** issues an ACL that is encrypted using the secret key of the host of the given service. It adds this additional information to the EUET. The ACL contains information, such as the CUID, requested SVID with permissions, and all other SVIDs that are accessible through the requested service with permissions, as shown in Fig. 4.7.

Cloud_server also issues a new tag to access the virtual service host (**v_service_host**). However, this tag structure is different from the one used earlier because it is used to identify **cloud_user** to **v_service_host**. The new tag considers timestamp T and the CUID, both encrypted by the secret key of **v_service_host**. In this case, **cloud_server** sends the encrypted ACL along with modified EUET in conjunction with newly generated $Tag_{\mathbf{v\_service\_host}}$ to **cloud_user**. Hence, the goal of allocating **cloud_user** to the requested service is achieved. At the end of this protocol, **cloud_user** has all the required information to access the requested service after a lightweight authentication process.

4.4 Host Session and Authentication Protocol

The main goal of the host session and authentication protocol is to authenticate **cloud_user** by the host of the requested service in a lightweight authentication process. In this protocol, **cloud_user** internally communicates with the required virtual service host. Consequently, **cloud_user** must be authenticated by **v_service_host** before accessing the requested service; however, this authentication is a lightweight authentication process to ensure that the requester is a legitimately authenticated **cloud_user**. As shown in Fig. 4.8, the service host lightweight authentication protocol functions as follows:

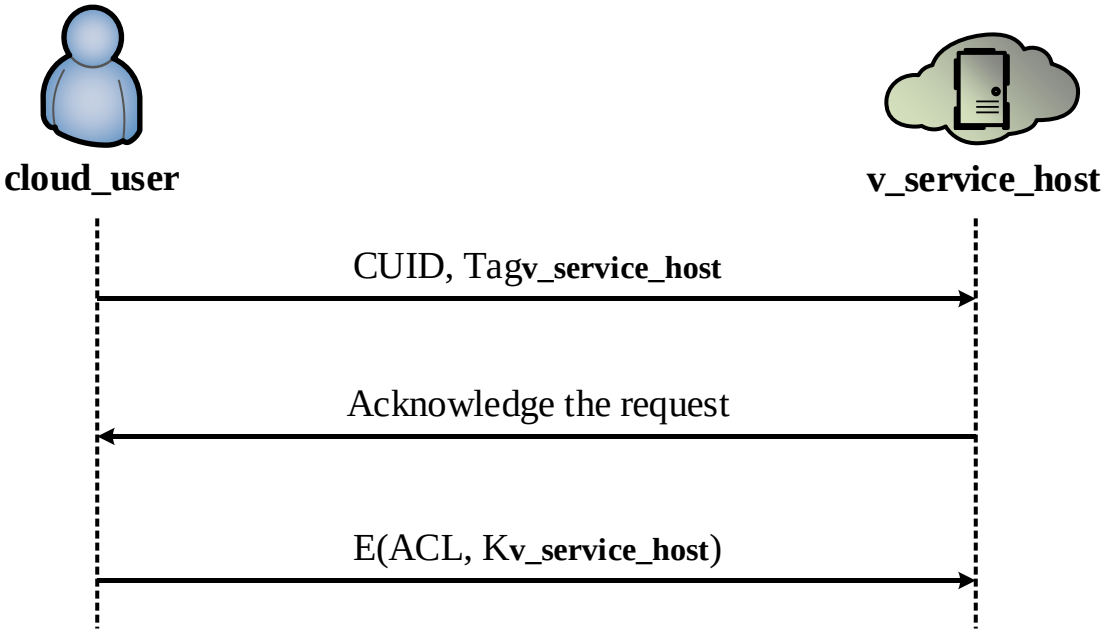

Figure 4.8: Service host lightweight authentication protocol in CSAM-2

Cloud_user sends a request that includes the CUID and $Tag_{v\_service\_host}$ to **v_service_host**.

V_service_host maintains its own lookup table based on the received CUID and $Tag_{v\_service\_host}$. **V_service_host** then validates **cloud_user** by decrypting the received tag and comparing the CUID in $Tag_{v\_service\_host}$ with the received CUID. This authentication is a lightweight process because **cloud_user** is already authenticated by **cloud_server**. Once the CUID is identified, **v_service_host** acknowledges the request. **Cloud_user** then sends the encrypted ACL to **v_service_host** to decrypt it. Next, **v_service_host** allows

cloud_user to access the requested service based on the registered information in the ACL. Hence, **cloud_user** is authenticated by the host of the requested service.

4.5 Discussion

A second module of a cloud-based secure authentication protocol suite is proposed in this chapter to add extra layers of protections to securely authenticate and identify a cloud user and prevent internal risks of DoS attacks. This protocol suite consists of four protocols to prevent the risks of DoS attacks in multiple cloud-computing service models. We have combined the EUET application technique, client puzzle problem, and deadlock avoidance algorithm to prevent security threats that allow attackers to perform cloud-based DoS attacks.

The proposed CSAM-2 protocol suite integrates several layers of protections. The protocol suite is designed to identify and authenticate legitimate user requests. Moreover, the protocol suite is developed to prevent external DoS attackers based on the theory of computational complexity. Therefore, a cloud-computing server can adjust the difficulty level of the client puzzle based on the sensitivity of the requested service(s). It thereby protects the cloud server from external DoS attackers. Additionally, the deadlock avoidance algorithm is implemented to allocate the requested service(s) to the authenticated cloud user with the ability to detect threats of internal DoS attacks. Furthermore, the CSAM-2 protocol suite is developed to enable the host of the requested service to authenticate the cloud user.

The proposed CSAM-2 protocol suite is practically applicable to different cloud computing service models. This is because the protocol is developed based on basic software and hardware components of both participants. The proposed CSAM-2 protocol is reliable on account of two key reasons. First, the processes of the protocol suite do not overload the cloud resources. Second, the protocol suite controls access to the cloud service(s) by implementing the deadlock problem with no cache process. Therefore, the cloud server does not allocate the cloud user to the requested service(s) if the number of maximum allowed concurrent accesses to these service(s) is achieved. Furthermore, the maximum number of allowed accesses to the cloud services is adaptive based on the properties of

these services. As a result, the CSAM-2 protocol suite is scalable because it can be expandable based on the size of the cloud-computing system without affecting the cloud computing resources.

Chapter 5

5 Security and Performance Validation

In this chapter, the achievement of the security requirements of the authentication protocol by both CSA protocol modules is verified. In addition, the performance of the protocols is assessed. The security validation includes the assessment of the CSA protocols via the Meadows cost-based model approach. The formal verification of the authentication processes of the protocols is performed via SVO logic. In terms of performance, the knapsack puzzle technique used in the protocol design is validated. Furthermore, the implementation of Google's OAuth 2.0 for web server applications is evaluated. Finally, the performance of highly computational processes of CSA protocols is evaluated. The assessment results are compared with the evaluation outcomes of Google's OAuth 2.0 implementation for web server applications.

5.1 Security Validation

In this section, both modules of CSA protocol suites are analyzed and assessed in terms of their security implementations. The assessment of the CSA protocols entails an evaluation of the protocol's efficiency against DoS attacks by applying a cost-based model approach. In addition, the security effectiveness of the proposed authentication protocols via SVO logic is analyzed. This analysis proves that the proposed authentication protocols in CSA achieve the authentication requirements of an authentication protocol for cloud-computing systems.

5.1.1 Security Validation of the CSA Protocol Suite via a Cost-Based Model Approach

The Meadows cost-based model is an approach to analyze the computation process of an authentication protocol when it comes to its vulnerability to DoS attacks (Meadows, 2001). This technique is designed to avoid DoS attacks through the authentication operation. This cost-based model relies on the costs of the exhausted resources of the protocol contributors. The cost-based model approach practically demonstrates the ability of the protocol to avoid DoS attacks. In this approach, the computation cost is described as the overall resource

consumption cost of the user and the server when they become involved in the authentication process. The cost is computed throughout the authentication process prior to the process of detecting a DoS attacker, who is then prevented from completing the authentication process. The user's total cost is the total cost of every single operation in the authentication process—from the user's component until the completion of the authentication process. Additionally, the servers' total costs are the total costs of every operation throughout the authentication process until either the user is authenticated as legitimate or the attacker is detected. The following categorizations are proposed by Meadows (Meadows, 2001) for the cost of an operation: expensive, medium, and inexpensive. The Meadows approach considers that the signature, a check signature, and the exponential operations that are executed throughout the authentication process are expensive. The decryption, encryption, and pre-calculated exponential value operations have medium costs. Every other operation is inexpensive.

As presented in Table 5.1, based on the cost-based model approach, the operation cost of **cloud_user** for the CSA protocol is categorized as expensive, particularly when **cloud_user** solves the puzzle. The other operations of **cloud_user** are listed in the medium or inexpensive categories. However, the maximum operation costs of **cloud_server**, including the pre-calculation and decryption operations, are in the medium category. As a result, the CSA protocol suite is an effective protocol against DoS attacks, in which the user consumption cost is higher than the consumption cost of the cloud service provider during the authentication process.

Table 5.1: Validation of the CSA protocol suite via a cost-based model approach

CSA protocol	Cloud_user		Cloud_server	
	Operation	**Cost Category**	**Operation**	**Cost Category**
Identification and authentication protocol in CSAM-1 and CSAM-2	Sends the initial request	Inexpensive	Replies directly to the request via secure hashing of the received values to obtain the puzzle element and asks the **cloud_user** for the UET	Inexpensive-Medium
	Solves the puzzle until the result is obtained. Then, sends the result and the UET to **cloud_server**	Expensive	Verifies the received elements. Encrypts the SK	Medium
	Decrypts the SK. Encrypts the timestamp	Medium	Decrypts the UET. Decrypts the timestamp	Medium
Service allocation protocol in CSAM-2	Sends the initial request.	Inexpensive	Verifies the request. Replies with a list of available services	Inexpensive
	Requests a service	Inexpensive	Considered a deadlock algorithm. Encrypts the ACL	Medium
Service host lightweight authentication protocol in CSAM-2	Sends the initial request	Inexpensive	Acknowledges the request	Inexpensive
	Sends the received encrypted ACL	Inexpensive	Decrypts the ACL	Medium

5.1.2 Security Validation and Formal Verification of the CSA Protocol Suite via SVO Logic

Logic-based formal approaches typically are used to analyze the security properties of cryptographic protocols. These approaches use notations and axioms to study the beliefs of protocol participants. The consequences of these beliefs are analyzed to determine whether the protocol's security goals have been achieved. SVO logic (P. F. Syverson & Van Oorschot, 1994) is our tool to analyze CSA protocols.

In the proposed CSA protocol suite, the authentication processes are performed in the last three steps of the adaptive-based identification and authentication protocol in both modules, CSAM-1 and CSAM-2, as shown in Fig. 5.1. Furthermore, the validation is performed in the service host authentication protocol in CSAM-2, as shown in Fig. 5.2 further below. In addition, Table 5.2 shows the notations and their descriptions that are used in the SVO logic.

Table 5.2: Notations of the SVO

Notation	Description
P believes X	*P* can take *X* as true
P received X	*P* has received a message containing *X*
P said X	*P* believes *X* when *P* sent it
P says X	*P* has said *X*
P has X	*X* is initially available to *P*, freshly generated by *P*, or received by *P*
P controls X	*P* has a jurisdiction on *X*
fresh(*X*)	*X* is fresh, and it has not been previously sent
$P \xleftrightarrow{k} Q$	*P* and *Q* communicate with each other by a good shared key, *k*
$PK_{\psi}(P, k)$	*k* is a public encryption key of *P*. Only *P* can read messages encrypted by *k*
$PK_{\sigma}(P, k)$	*k* is a public signature key of *P*. Key *k* verifies that the messages signed by the corresponding private key k^{-1} are from *P*
$PK_{\delta}(P, k)$	*k* is a public key agreement of *P*. A Diffie-Hellman key formed with *k* is shared with *P*
$\{X\}_k$	*X* is encrypted under key *k*
$\lfloor X \rfloor_k$	*X* is signed with key *k*
$<X>_{*P}$	Received message *X* is unrecognized by *P*

The two inference rules of SVO are outlined below.

Modus Ponens:

$$\frac{\varphi \quad \varphi \rightarrow \psi}{\psi}$$

The rule is that whenever the instance φ and the instance $\varphi \rightarrow \psi$ (i.e. φ implies ψ) appear by themselves on lines of a proof then ψ can validly be placed on a subsequent line.

Necessitation:

$$\frac{\vdash \varphi}{\vdash P\ believes\ \varphi}$$

where $\Gamma \vdash \varphi$ means that φ can be derived from the set of formulas, Γ. Using the above rules, "$\vdash \varphi$" means that φ is a theorem.

The axioms of SVO (P. F. Syverson & Van Oorschot, 1994) are outlined below. We assume that the reader is got familiar with the notations that described in Table 5.2.

Belief Axioms:

A1. $(P\ believes\ \varphi \wedge P\ believes\ (\varphi \rightarrow \psi)) \rightarrow P\ believes\ \psi$

A2. $P\ believes\ \varphi \rightarrow \varphi$

A3. $P\ believes\ \varphi \rightarrow P\ believes\ (P\ believes\ \varphi)$

Source Association Axioms:

A4. $(P \overset{k}{\leftrightarrow} Q \wedge R\ received\ \{X\ from\ Q\}_k) \rightarrow (Q\ said\ X \wedge Q\ has\ X)$

A5. $(PK_\sigma(Q,k) \wedge R\ received\ X \wedge SV(X, k, Y)) \rightarrow Q$ said Y

Key Agreement Axioms:

A6. $(PK_\delta(P, k_P) \wedge PK_\delta(Q, k_Q)) \rightarrow P \overset{F_0(k_P,k_Q)}{\longleftrightarrow} Q$

A7. $\varphi \equiv \varphi\ [F_0(k, k') / F_0(k', k)]$

Receiving Axioms:

A8. $P\ received\ (X_1, \ldots X_n) \rightarrow P\ received\ X_i$ for $i = 1, \ldots, n$

A9. $(P \text{ received } \{X\}_{k+} \wedge P \text{ has } k^{-}) \rightarrow P \text{ received } X$

A10. $(P \text{ received } \lfloor X \rfloor_k) \rightarrow P \text{ received } X$

Possession Axioms:

A11. $P \text{ received } X \rightarrow P \text{ has } X$

A12. $P \text{ has } (X_1, \ldots X_n) \rightarrow P \text{ has } X_i$ for $i = 1, \ldots, n$

A13. $(P \text{ has } X_1 \wedge \ldots \wedge P \text{ has } X_n) \rightarrow P \text{ has } F(X_1, \ldots X_n)$

Comprehension Axiom:

A14. $P \text{ believes } (P \text{ has } F(X)) \rightarrow P \text{ believes } (P \text{ has } X)$

Saying Axioms:

A15. $P \text{ said } (X_1, \ldots X_n) \rightarrow P \text{ said } X_i \wedge P \text{ has } X_i$ for $i=1,\ldots, n$

A16. $P \text{ says } (X_1, \ldots, X_n) \rightarrow (P \text{ said } (X_1, \ldots X_n) \wedge P \text{ says } X_i)$ for $i = 1, \ldots, n$

Freshness Axioms:

A17. $\text{fresh } (X_i) \rightarrow \text{fresh } (X_1, \ldots, X_n)$ for $i = 1, \ldots, n$

A18. $\text{fresh } (X_1, \ldots, X_n) \rightarrow \text{fresh } F(X_1, \ldots, X_n)$

Jurisdiction and Nonce-Verification Axioms:

A19. $(P \text{ controls } \varphi \wedge P \text{ says } \varphi) \rightarrow \varphi$

A20. $(\text{fresh } (X) \wedge P \text{ said } X) \rightarrow P \text{ says } X$

Symmetric Goodness Axiom:

A21. $P \overset{k}{\leftrightarrow} Q \equiv Q \overset{k}{\leftrightarrow} P$

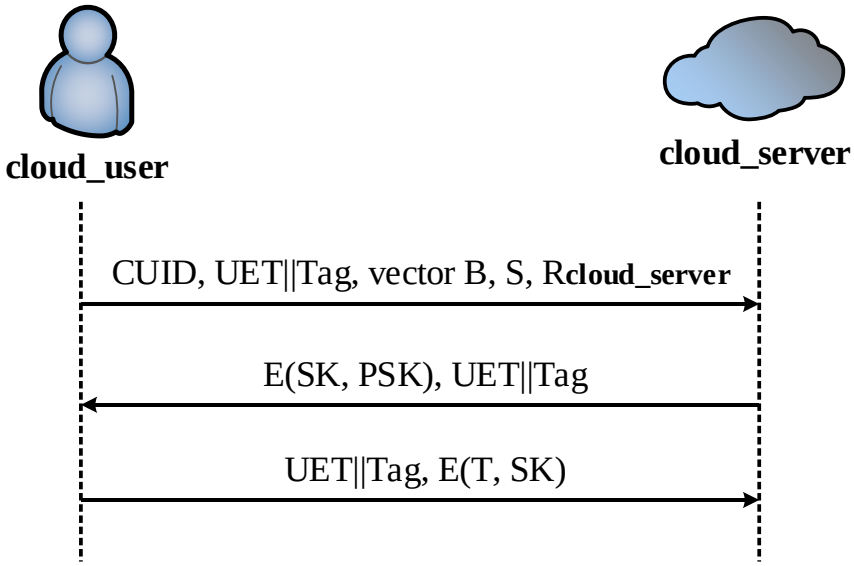

Figure 5.1: Authentication processes of CSAM-1 and CSAM-2

To formally analyze the authentication process in CSAM-1 and CSAM-2 via SVO logic, as shown in Fig. 5.1, it is important to identify the goal of the protocol. In this protocol, the goal is stated as follows:

cloud_server authenticates the ***cloud_user***, so that

cloud_server *believes* (***cloud_user*** *says T*)

where *PSK* is a pre-shared key, and *SK* is a session key.

The protocol analysis is presented in five steps that are the initial state assumption, received message assumption, comprehension assumption, interpretation assumption, and derivation.

1) Initial state assumption

The initial state assumption includes all initial statuses of the protocol.

I1. ***cloud_user*** *believes* ***cloud_user*** $\overset{PSK}{\longleftrightarrow}$ ***cloud_server***

I2. ***cloud_server*** *believes* ***cloud_user*** $\overset{PSK}{\longleftrightarrow}$ ***cloud_server***

I3. ***cloud_user*** *believes* (***cloud_server*** *controls SK*)

I4. ***cloud_server*** *believes fresh*(*SK*)

I5. ***cloud_user*** *believes fresh*(*SK*)

2) Received message assumption

The received message assumption step indicates what messages each party receives.

R1. ***cloud_user*** *received* $\{SK\}_{PSK}$

R2. ***cloud_server*** *received* $\{T\}_{SK}$

3) Comprehension assumption

This step states what receivers believe and indicates what parts of the received message are unknown.

C1. ***cloud_user*** *believes* (***cloud_user*** *received* $\{SK\}_{PSK}$)

C2. ***cloud_server*** *believes* (***cloud_server*** *received* $\{T\}_{<SK>*cloud\_server}$)

4) Interpretation assumption

The interpretation assumption step shows what the sender intends by sending the message.

P1. ***cloud_user*** *believes* (***cloud_user*** *received* $\{SK\}_{PSK} \rightarrow$ ***cloud_user*** *received* $\{SK \wedge fresh(SK)\}_{PSK}$)

5) Derivation

The derivation step derives the analysis goal by the previous assumptions.

D1. ***cloud_user*** *believes* ***cloud_user*** *received* $\{SK \wedge fresh(SK)\}_{PSK}$

By applying Modus Ponens, C1, P1.

D2. ***cloud_user*** *believes* (***cloud_server*** *said SK* ^ ***cloud_server*** *has SK*)

By applying the Source Association (A4), D1, I1, I2, and Belief Axiom.

D3. $\boldsymbol{cloud\_user} \overset{SK}{\longleftrightarrow} \boldsymbol{cloud\_server}$

By applying the Receiving (A9), R1, D2, and Belief Axioms.

D3 shows that ***cloud_user*** and ***cloud_server*** communicate with each other by a good shared key (*SK*).

D4. ***cloud_server*** *believes* (***cloud_user*** *said T* ^ ***cloud_user*** *has T*)

By applying the Source Association (A4), C2, D3 and Belief Axioms.

D5. ***cloud_server*** *believes* (***cloud_user*** *said T*)

By applying the Saying (A15), D4, and Belief Axioms.

D6. ***cloud_server*** *believes* (***cloud_user*** *says T*)

By applying the Jurisdiction (A20), I5, D5 and Belief Axioms.

D6 shows that our analysis goal, which aims to prove that ***cloud_server*** authenticates ***cloud_user***, has been achieved by applying the rules of SVO logic.

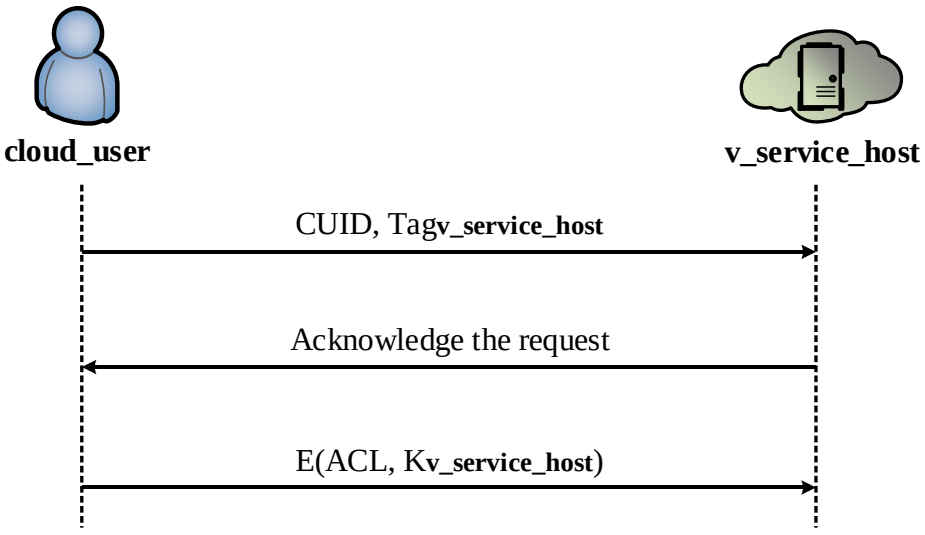

Figure 5.2: Additional authentication processes of CSAM-2

To formally analyze the additional authentication process in CSAM-2 via SVO logic, as shown in Fig. 5.2, the goal of the protocol is outlined below.

v_service_host authenticates ***cloud_user*** by believing in the received message from ***cloud_user***, so that:

v_service_host *believes* (***cloud_server*** *says ACL*)

where *K* is a $K_{\textbf{v\_service\_host}}$.

Once again, the protocol analysis is presented in five steps that are the initial state assumption, received message assumption, comprehension assumption, interpretation assumption, and derivation.

1) Initial state assumption

The initial state assumption includes all initial statuses of the protocol.

I1. ***v_service_host*** *believes* (***cloud_server*** $\overset{K}{\longleftrightarrow}$ ***v_service_host***)

I2. ***cloud_server*** *believes fresh*(*ACL*)

2) Received message assumption

The received message assumption step indicates what messages each party receives.

R1. ***v_service_host*** *received* $\{ACL\}_K$

3) Comprehension assumption

This step states what receivers believe and indicates what parts of the received message are unknown.

C1. ***v_service_host*** *believes* (***v_service_host*** *received* $\{ACL\}_{<K>*\textbf{cloud\_user}}$)

4) Interpretation assumption

The interpretation assumption step shows what the sender intends by sending the message.

P1. ***v_service_host*** *believes* (***v_service_host*** *received* $\{ACL\}_{<K>*\textbf{cloud\_user}} \rightarrow$ ***v_service_host*** *received* $\{ACL \wedge fresh(ACL)\}_K$)

5) Derivation

The derivation step derives the analysis goal by the previous assumptions.

D1. ***v_service_host*** *believes* ***v_service_host*** *received* $\{ACL \wedge fresh(ACL)\}_K$

By applying Modus Ponens, C1, P1.

D2. ***v_service_host*** *believes* (***cloud_server*** *said ACL* ^ ***cloud_server*** *has ACL*)

By applying the Source Association (A4), D1, I1 and Belief Axioms.

D3. ***v_service_host*** *believes* (***cloud_server*** *said ACL*)

By applying the Saying (A15), D2 and Belief Axioms.

D4. ***v_service_host*** *believes* (***cloud_server*** *says ACL*)

By applying the Jurisdiction (A20), I2, D1, D3 and Belief Axioms.

D4 shows that our analysis goal, which is intended to prove that ***v_service_host*** authenticates ***cloud_user***, has been achieved by applying the rules of SVO logic.

5.2 Performance Validation

In this section, the performance of the knapsack puzzle problem is analyzed via dynamic programming. This analysis is performed to determine the difficulty levels of the puzzle so that the challenge technique can be adjusted on the basis of the analysis results. Furthermore, the implementation of Google's OAuth 2.0 for web server applications is evaluated. This indicates that the implementation of Google's OAuth 2.0 protocol may lead to a security flaw that exploits low- to medium-size web servers. This threat may occur by exhausting the storage resources of the web server and by rendering its applications unavailable. Finally, the effects of the high computation processes of the CSA protocols are evaluated. These processes include the process of generating random number,

encrypting plaintext by a well-known key, decrypting ciphertext by a well-known key, and the hashing process on the cloud server. A comparative analysis of the evaluation results and those of Google's OAuth 2.0 implementation for web server applications is additionally presented.

5.2.1 Knapsack Puzzle Performance Validation

An experiment was conducted to analyze the time complexity of the subset sum (knapsack) problem. The subset sum can be briefly described as follows. Assume a set of positive integers A of size n and a positive integer value S. Let us determine whether any non-empty subset of size m adds up to S. For example, let A be (13, 54, 28, 73, 3, 36), n = 6, and S = 89. It is obvious that for the subset (13, 73, 3), m = 3 solves the problem because their summation is equal to 89. In other words, let us find a binary vector B such that A . B = S solves the problem. In this example, B is the vector (1, 0, 0, 1, 1, 0); hence, A . B = 13 + 73 + 3, which is 89.

It should be noted that, in the identification and authentication protocols, vector B is generated as the output of the hash function (see Chapters 3.2 and 4.2).

It was mentioned in Chapter 3.2 that the values of n and m are key factors that play a significant role in the complexity of the subset sum problem. In this experiment, different values of n and m are chosen for which the time complexity of the subset sum problem is analyzed. The values of n are 128, 256, and 512, while the values of m range from 1 to 64. The dynamic programming algorithm is used to solve the puzzle; it is coded in C# and runs on a quad-core desktop computer with the Windows 8 64-bit operating system, a Core i7-4770 CPU running at 3.4 GHz, and 32 GB of RAM.

Our experiment indicates that the algorithm solves the puzzle in less than 8 seconds with n = 128 and for all values of m. When n = 256, and for all values of m, the algorithm solves the puzzle in less than 12 seconds. Finally, when n = 512 and for all values of m, the algorithm solves the puzzle in less than 24 seconds. The detailed execution times when n = 512 are shown by the graph in Fig. 5.3. The graph indicates that the puzzle is solved in approximately 10 seconds when m is between 25 and 30. When m is between 55 and 60,

the puzzle is solved in approximately 20 seconds of the execution time. It is also to be noted that the algorithm hangs on account of the full consumption of system memory when n = 512 and m is chosen to be higher than 64; i.e., the system resources are exhausted.

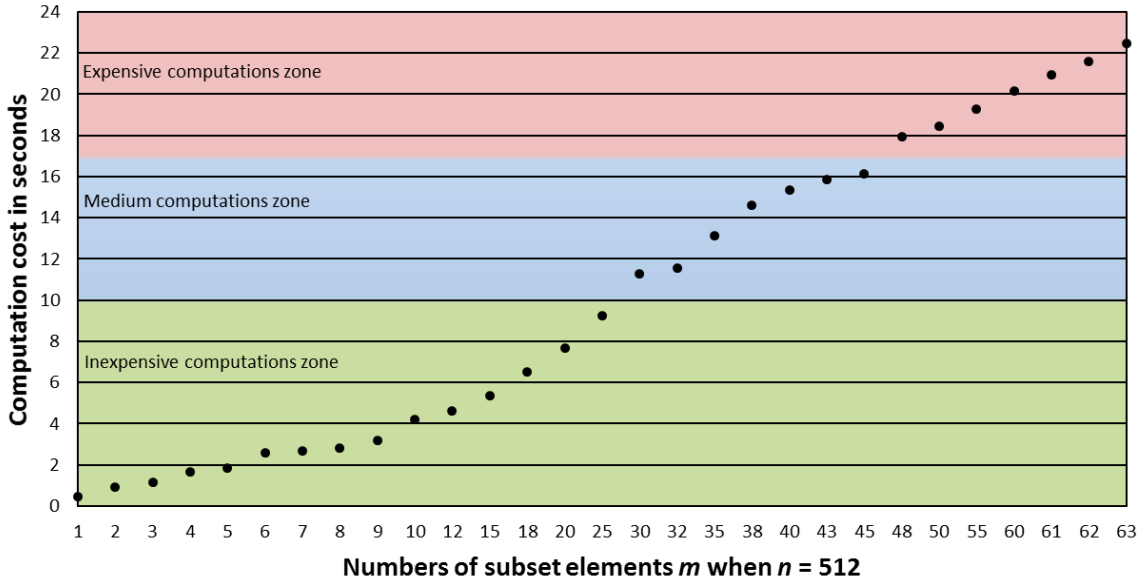

Figure 5.3: Response times of the requester (in seconds) with various numbers of combination items

Based on these results and the corresponding system resource consumption, the computation costs of solving the subset sum problem can be categorized into three main categories—inexpensive, medium and expensive—as shown in Fig. 5.3.

According to the findings of the surveys conducted by Nielsen (Nielsen, 1993), 15 to 20 seconds is an acceptable response time for maintaining the user's attention on a given application. Therefore, choosing n = 512 and m = 55 is a good configuration of the subset sum problem. This configuration enables the legitimate user to perform expensive computations that are acceptable in terms of the system response time (as per the Nielsen study) especially in terms of his/her initial request.

It is worth mentioning that other researchers (Tritilanunt, Boyd, Foo, & Nieto, 2007) have shown that the L3 algorithm developed by Lenstra et al. (Lenstra, Lenstra Jr., & Lovász, 1982) can solve the subset sum problem of n = 100 and m = 80 in 2,700 seconds. These

researchers have contended that this configuration makes the puzzle difficult to solve. However, this is not true, especially when the dynamic programming algorithm is instead used. Therefore, the L3 algorithm is not recommended for use in the CSA protocol suite.

It seems reasonable to conclude that the difficulty level of the subset sum (knapsack) problem, and, hence, the cost-based approach, can be adaptively realized by adjusting the values of n and m.

5.2.2 Performance Evaluation of the Google OAuth 2.0 Protocol Implementation

The data storage process in the authentication protocol can cause security breaches in the web servers. That is, it can be vulnerable to security threats, such as DoS attacks. In the OAuth 2.0 implementation described in Chapter 2.3.2, web servers store refresh tokens in their storage devices. They employ these tokens unless the individual user asks for them to be revoked. In this case, the web server storage capacity may be exhausted by many requests asking for a refresh token to be stored. As a result, the web server may be unable to execute legitimate requests, even though it strictly provides only one refresh token to each individual user. Therefore, this experiment evaluates the risk of implementing Google's OAuth 2.0 protocol for web server applications on cloud-based web servers that have limited resources.

To evaluate the risk of storing data during the authentication process in a cloud-based web server, an experiment was conducted using the GreenCloud simulator tool developed by Kliazovich et al. (Kliazovich, Bouvry, & Khan, 2012) and based on the NS2 network simulator tool (Fall & Varadhan, 2007).

In the experiment, the actual size of the refresh tokens of several vendors was first determined. Yahoo's documentation for OAuth 2.0 for web server applications (Google Developers, 2014) states that the approximate size of the refresh token is 52 bytes. Both Twitter's (Twitter, 2015) and Google's documentation (Yahoo, 2015) for OAuth 2.0 specify approximately 42 bytes for the refresh token. As a result, the average size of the refresh token in this experiment was set to 48 bytes.

In this experiment, the main server was configured to manage a single VM as a web server. The resources of the main server are typically shared by other VMs. Therefore, the VM's specifications are generally a quad-core CPU processor, 4 GB of RAM, and a 100-GB hard disk drive. A reasonable number of stored refresh tokens for this VM is 1,000,000 stored refresh tokens out of 1,500,000 service users, as in a real business case (LoginRadius, n.d.).

This experiment evaluated the performance of the VM, specifically its load, memory load, and storage load. It is supposed that, if execution of the VM's tasks requires over 50% of the load, the resources of the VM are being exhausted.

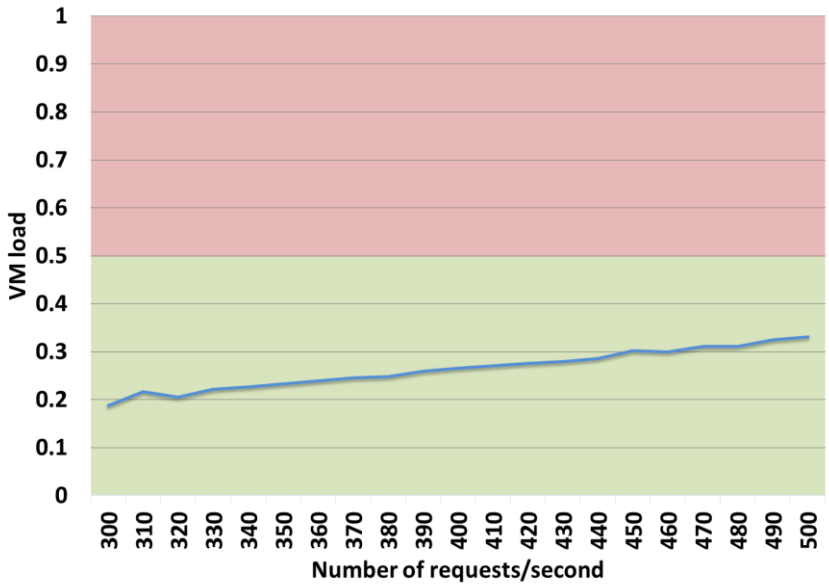

Figure 5.4: VM load with multiple requests per second

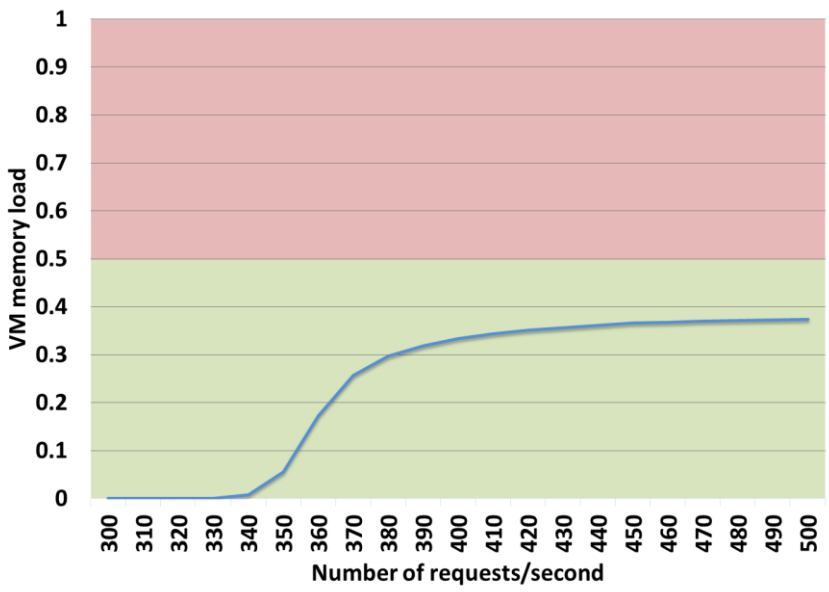

Figure 5.5: VM memory load with multiple requests per second

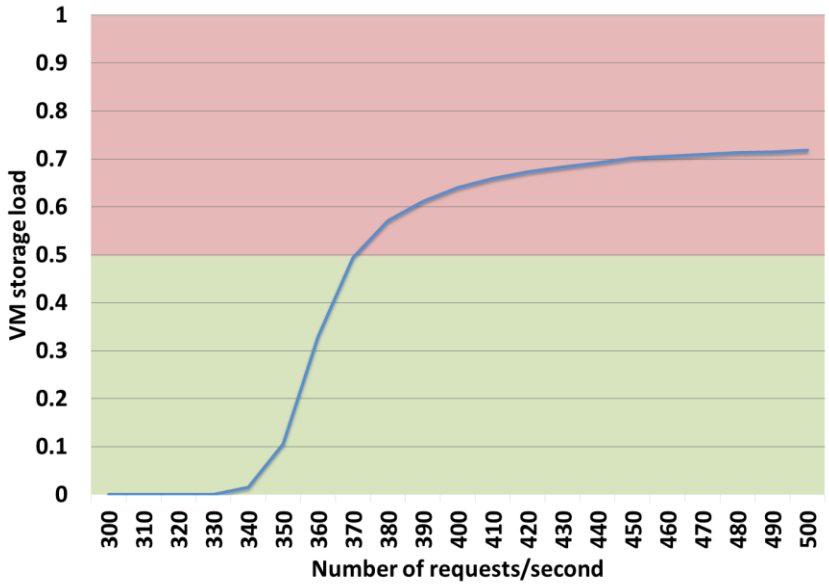

Figure 5.6: VM storage load with multiple requests per second

Figs. 5.4, 5.5, and 5.6 present the experimental results for the VM's load, memory load, and storage load, respectively. It is noted that the VM's storage load begins to noticeably increase at approximately 300 to 500 requests per second. Real web servers can execute up

to 870 to 1,230 requests per second in extreme situations, depending on the specifications of the web server (Parallels, 2010). It should also be noted that the storage size in this experiment was expandable because it depended on two factors: the number of stored refresh tokens, and the size of the refresh token.

The experimental results indicate that storing data during the user authentication process leads to security flaws in the web servers that store these data. Therefore, the CSA protocol suite avoids storing new records during the authentication processes.

To validate the effectiveness of the CSA protocol suite against risks of DoS attacks during the authentication processes, Section 5.2.3 presents the performance of the most expensive computation processes of CSA protocols on the cloud server simultaneously performed with multiple tasks.

5.2.3 Performance Validation of the CSA Protocol Suite

Based on our proposed CSA protocol suite design, the cloud server must proceed with many tasks. This experiment evaluated the effect of the most expensive computation processes of the CSA protocol, including generating random number (consider Linux-based random number generation), encrypting plaintext by a well-known key (consider AES256 encryption), decrypting ciphertext by a well-known key (consider AES256 decryption), and the hashing process (consider SHA2-512) on the cloud server. This experiment simulated these tasks using the GreenCloud simulation tool by configuring the cloud-server specification to be similar to the specification of a virtual web server in our first experiment. This similarity served to compare the performance of our proposed CSA protocol with the currently used cloud-computing authorization protocol. In this experiment, the effects of the most computationally expensive processes of the CSA protocol on VM load, memory load, and storage load when executing 300 to 500 requests per second were evaluated.

To obtain the parameters of the CSA protocol processes (“tasks” in GreenCloud), and to use them in the simulation tool, a real VM with four cores, 4 GB of RAM, and a 100-GB hard disk drive was implemented. A Linux operating system was installed on this VM to

benchmark the tasks and obtain their parameters. The task parameters in the simulation tool included the task's million instructions per second (MIPS), input size, output size, and storage size. To measure the task's MIPS, Linux provides a CPU speed metric called BogoMIPS. Therefore, after the task was executed in Linux, the MIPS of the task was computed by multiplying the used CPU BogoMIPS by the task's execution time (Pacini, Ribero, Mateos, Mirasso, & Garino, 2012). From that point, the information of the task's file size before and after execution was gathered. Finally, the size of the task's storage that was used during the task execution, such as the key size of the AES-256 encryption task, was obtained.

After executing the four tasks in Linux, the task parameters were computed, as shown in Table 5.3. These collected parameters were then used in the GreenCloud simulation tool.

Table 5.3: Task parameters obtained from the Linux benchmark experiment

Task type	AES-256 encryption	AES-256 decryption	SHA-512 hashing	Generating random number
MIPS	3,989	3,826	3,663	3,582
Input size	10,000 bytes	10,016 bytes	10,000 bytes	10 bytes
Output size	10,016 bytes	10,000 bytes	149 bytes	5,121 bytes
Storage size	48 bytes	48 bytes	10 bytes	10 bytes

The charts in Figs. 5.7 to 5.18 present our experimental results after simulating the four different tasks: AES-256 encryption, AES-256 decryption, SHA2-512 hashing, and random number generation.

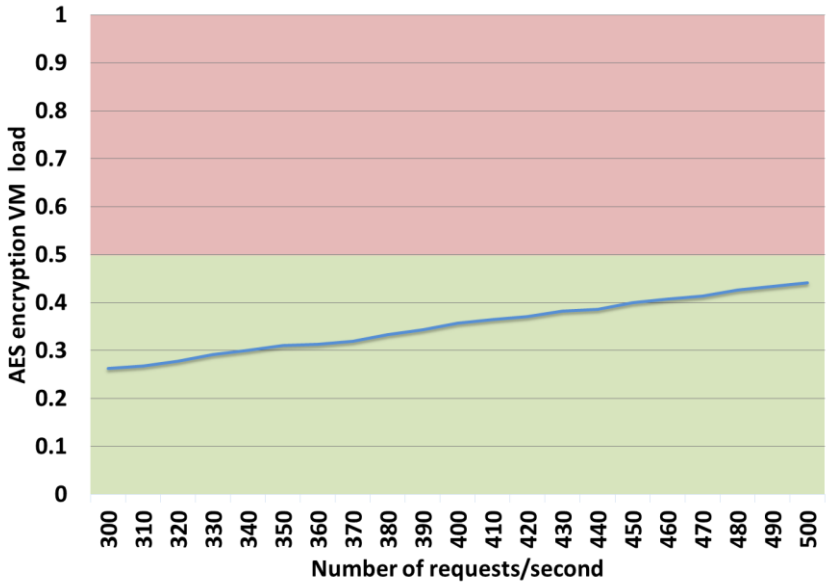

Figure 5.7: AES encryption VM load

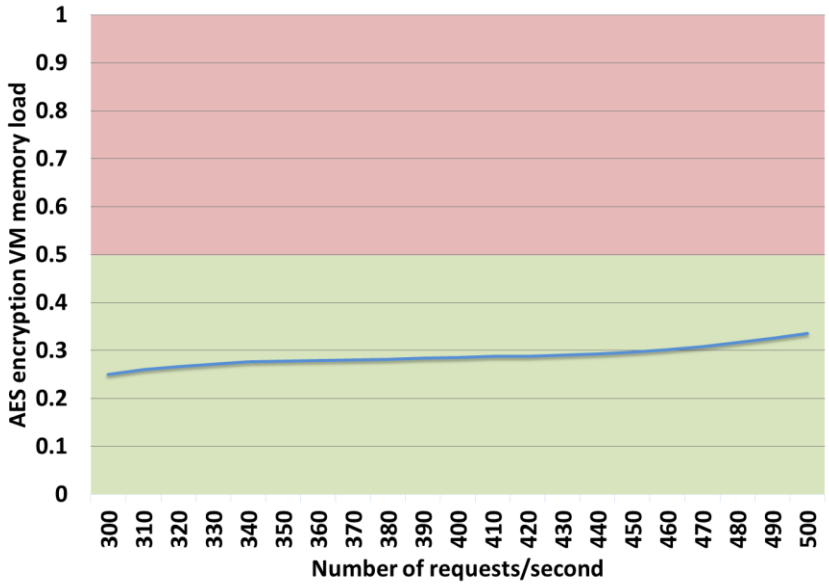

Figure 5.8: AES encryption VM memory load

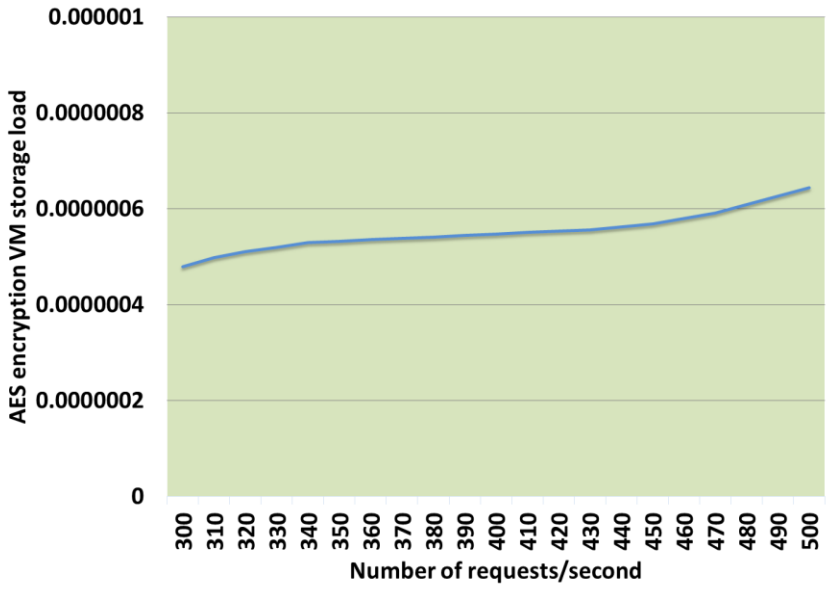

Figure 5.9: AES encryption VM storage load

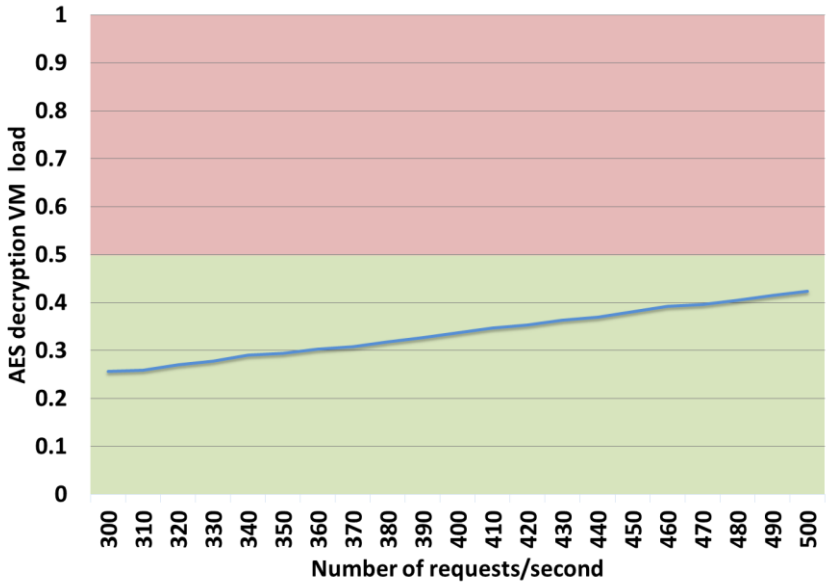

Figure 5.10: AES decryption VM load

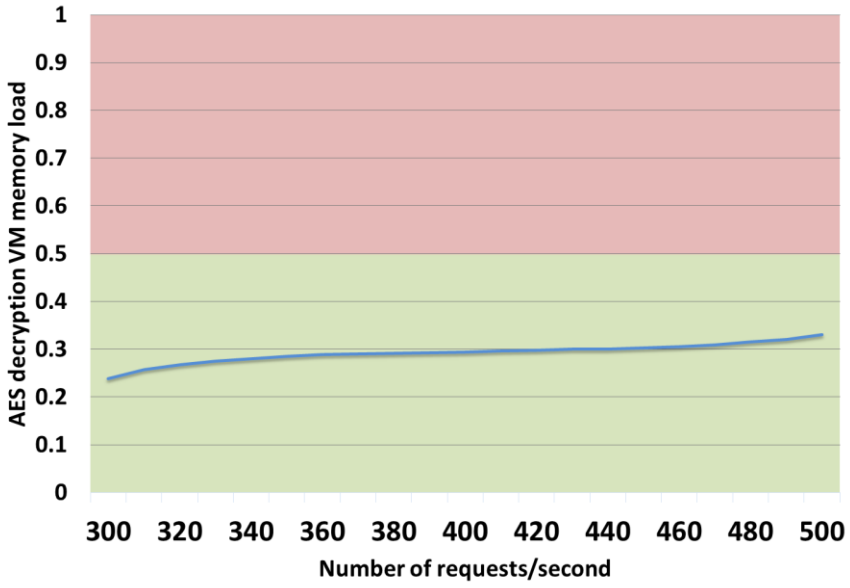

Figure 5.11: AES decryption VM memory load

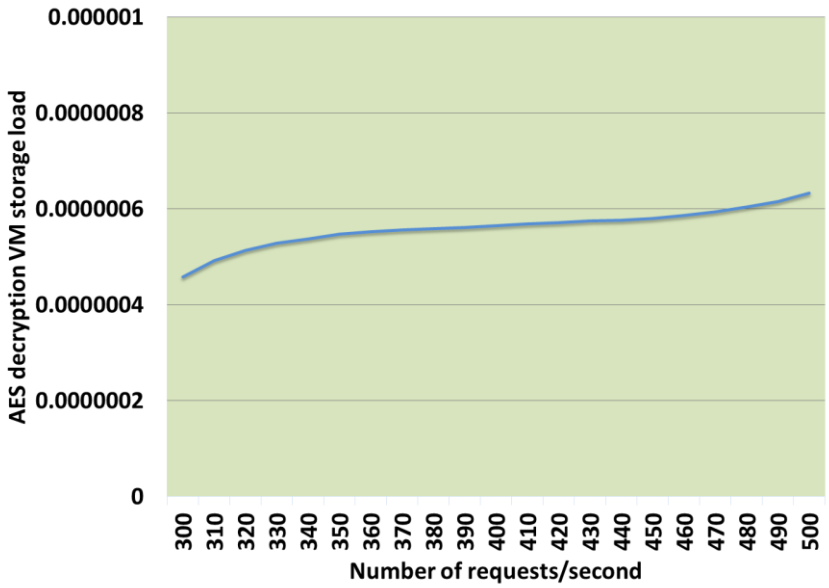

Figure 5.12: AES decryption VM storage load

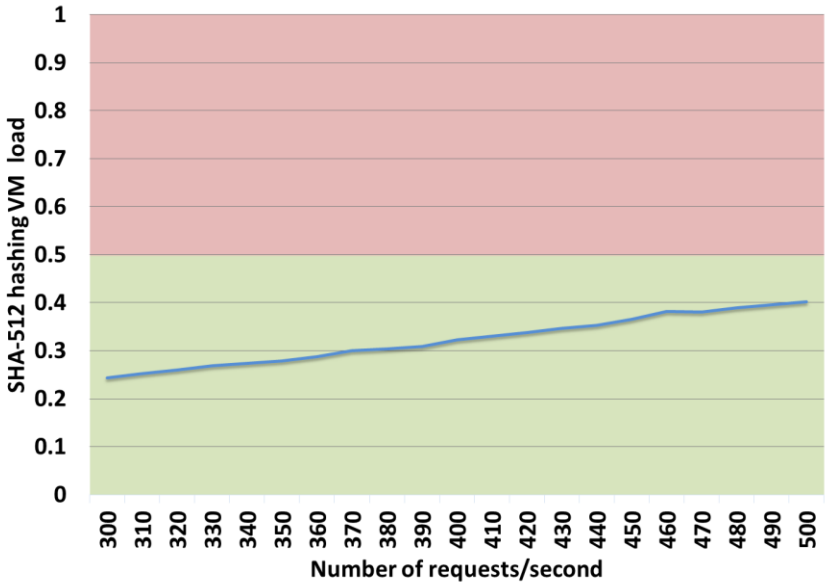

Figure 5.13: SHA-512 hashing VM load

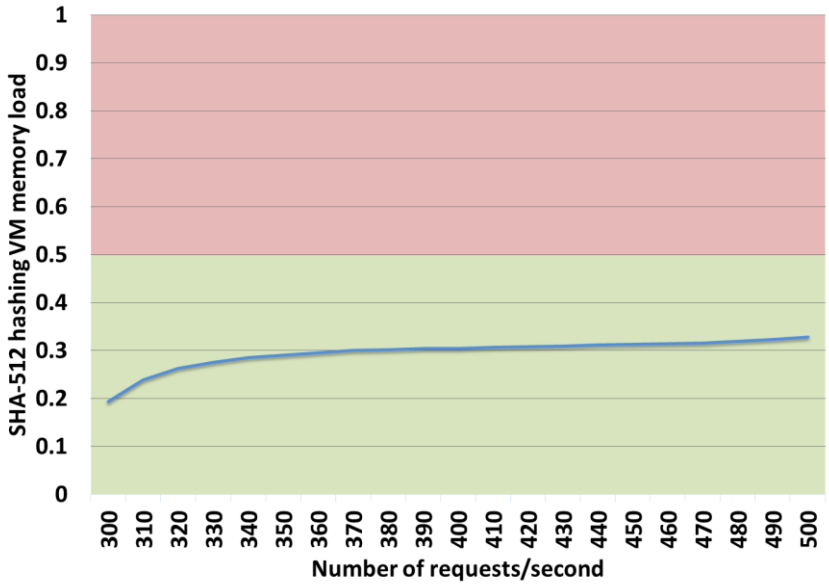

Figure 5.14: SHA-512 hashing VM memory load

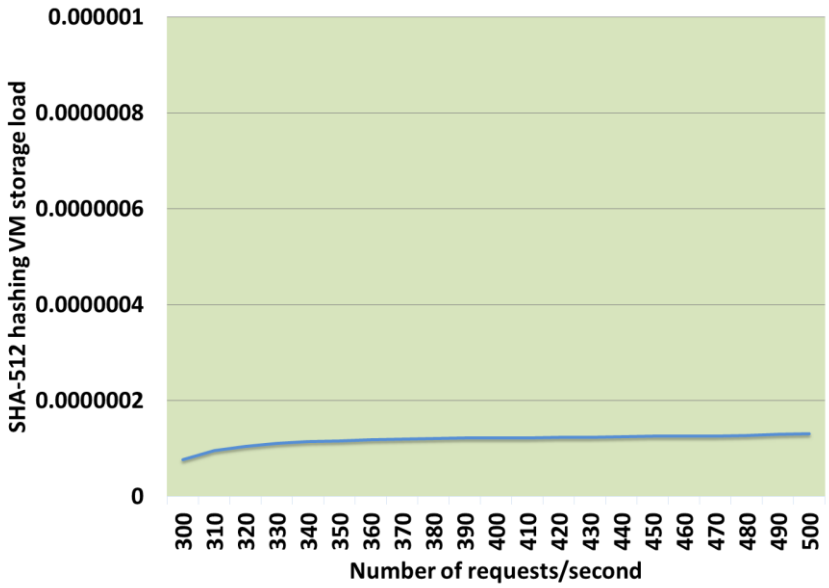

Figure 5.15: SHA-512 hashing VM storage load

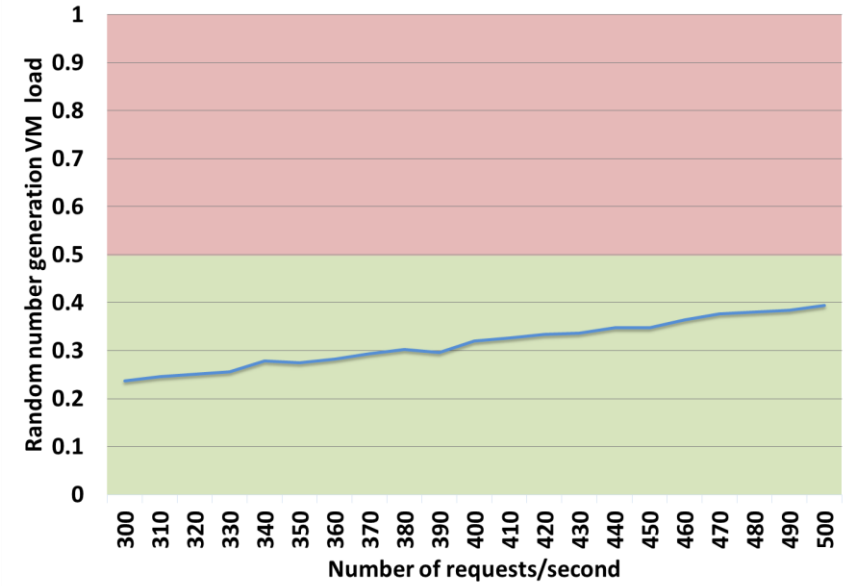

Figure 5.16: Generating random number VM load

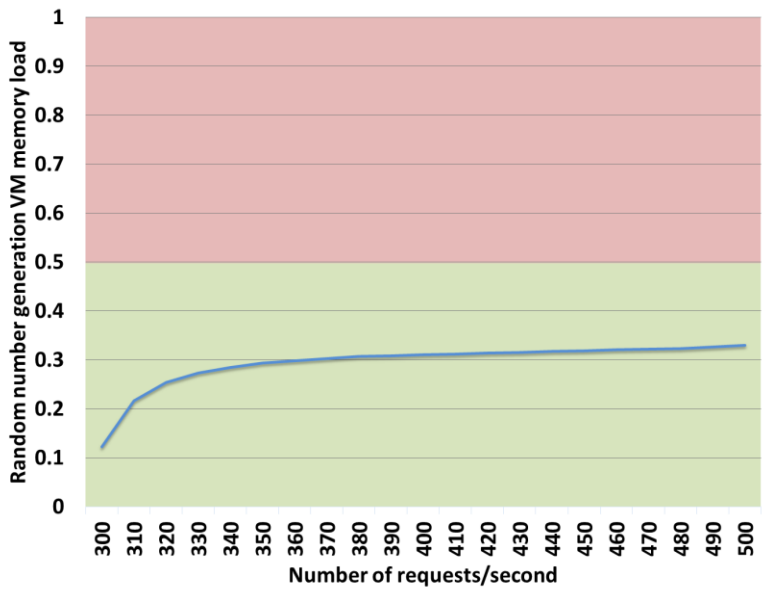

Figure 5.17: Generating random number VM memory load

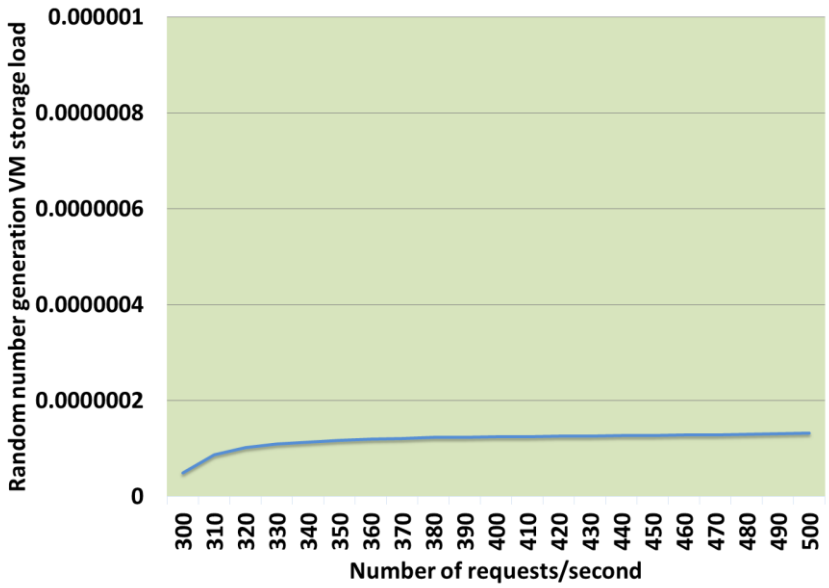

Figure 5.18: Generating random number VM storage load

From that point, the effect of the computationally expensive tasks of our proposed CSA protocol on the VM were compared to the effects of implementing the OAuth2.0 authentication protocol for the web server tasks simulated in our first experiment.

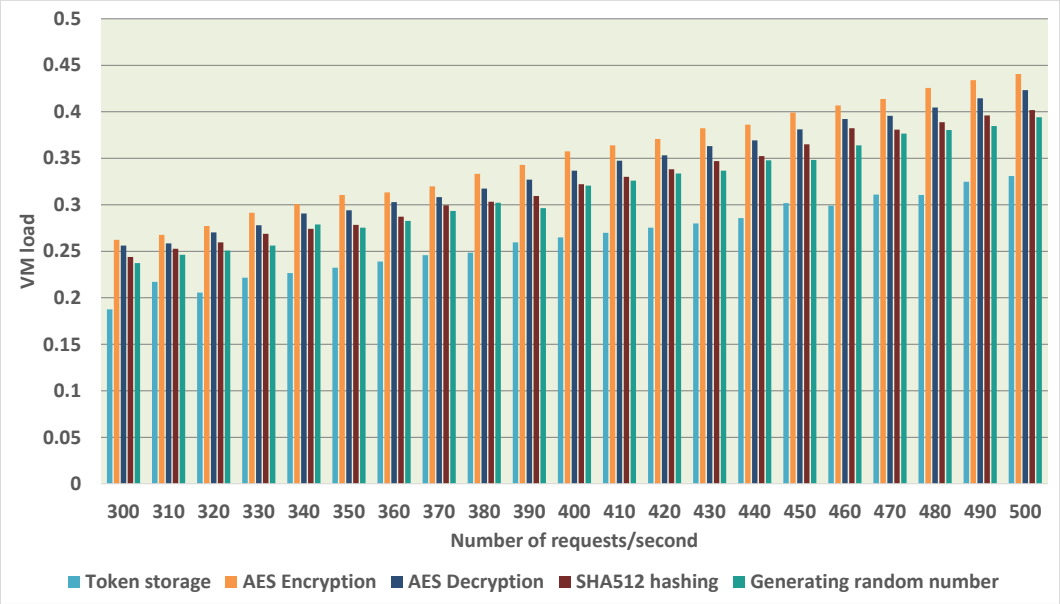

Figure 5.19: Comparison of VM load (the same graph is represented in tabular form in appendix A – Table A.1)

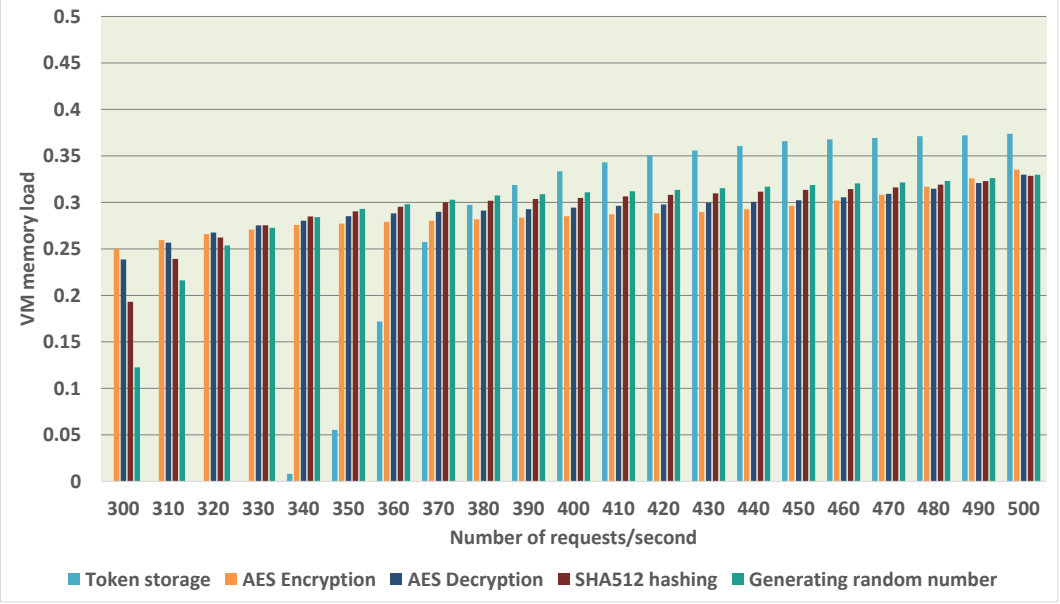

Figure 5.20: Comparison of VM memory load (the same graph is represented in tabular form in appendix A – Table A.2)

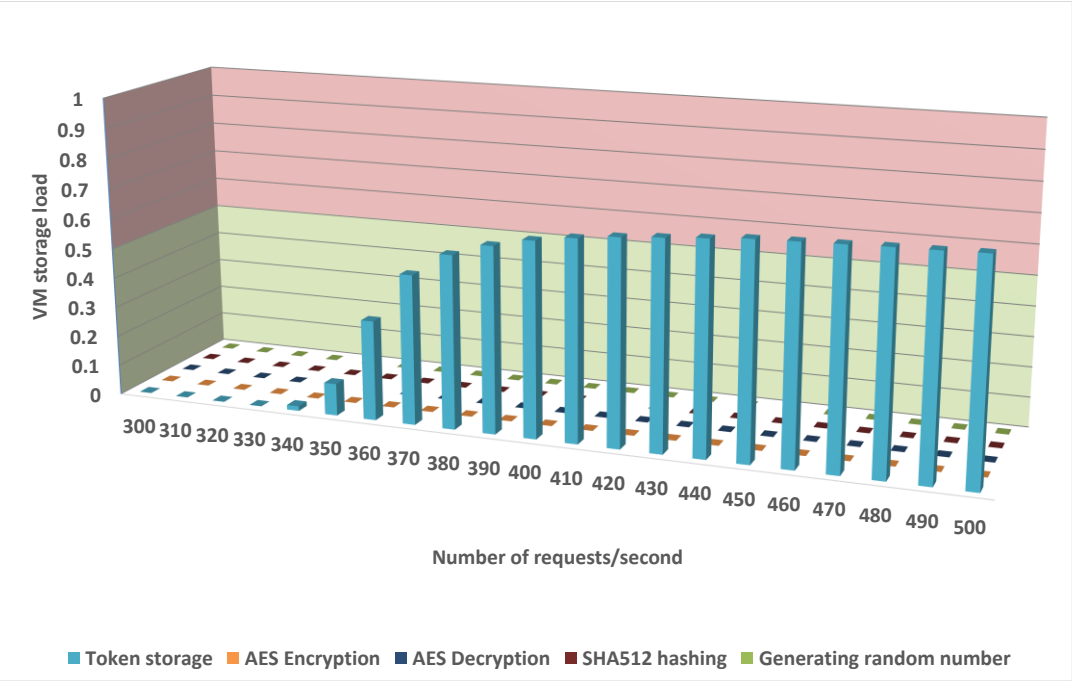

Figure 5.21: Comparison of VM storage load (the same graph is represented in tabular form in appendix A – Table A.3)

As shown in Figs. 5.19, 5.20, and 5.21, significant differences existed in the VM storage load of the proposed CSA protocol tasks compared with the simulated implementation of OAuth2.0. These differences were due to the immense amount of stored tokens in the web server that were used in the authentication process. Therefore, the web server could be vulnerable to DoS attacks on account of its storing of data during the authentication processes; this issue was avoided in the proposed CSA protocol design.

Chapter 6

6 Conclusions and Future Work

The use of software systems in a cloud-computing environment is increasingly common. DoS attacks are currently a major threat to the availability of cloud services. For each defense mechanism that has been developed against a DoS attack, an improved attack appears. The use of defense mechanisms from conventional networks to prevent DoS attacks in cloud-computing systems is not always efficient. Therefore, a cloud-based secure authentication (CSA) protocol suite for different cloud-computing deployment models was herein proposed. The authentication protocol identifies cloud users and securely authenticates them. Furthermore, it functions as a strong shield against risks of external and internal DoS attacks.

6.1 Summary of Contributions

The main goal of this thesis is designing and developing a novel cloud-based authentication protocol suite to securely authenticate the cloud user and to prevent risks of external and internal DoS attacks. The objectives that support our goal are achieved through the following tasks:

1. **Investigate DoS attacks in conventional networks and identify DoS attacks in cloud-computing systems.**

    - The risks of DoS attacks were investigated. In addition, existing types of DoS attacks in conventional network systems and their current defense mechanisms were investigated (Chapter 2).

    - The ability of cloud-computing systems to detect DoS attacks was also examined. Furthermore, we identified a taxonomy of existing DoS attacks and defenses on cloud-computing systems (Chapter 2).

2. **Investigate, propose, and validate a protocol suite that defends against external DoS attacks.**

- An investigation of the risks of DoS attacks in authentication protocols was presented (Chapter 2).

- The first module of a developed cloud-based secure authentication protocol (CSAM-1) that works against external cloud-based DoS attacks was proposed. This protocol can be implemented on private and community deployment models of the cloud system. To avoid security flaws that can produce DoS vulnerabilities, multiple techniques—including the client puzzle problem and unique encryption text (UET) application—were integrated into the design of the proposed authentication protocol. The CSAM-1 protocol suite depends on the computational complexity theory; therefore, the complexity level of the puzzle solution is adjustable. This theory was implemented in the protocol design to minimize the computation costs of cloud resources during the identification process. Moreover, it makes the computation costs of the cloud user's resources adaptable based on the sensitivity of the requested service. Therefore, the device's resources of the DoS attacker are involved in high computation processes when the attacker attempts to initiate a massive number of attacking requests. The CSAM-1 protocol suite was developed based on basic hardware and software requirements for cloud users and servers. Thus, the protocol is practically applicable in SaaS and PaaS service models (Chapter 3).

- The results of a validation indicated that the CSAM-1 protocol is a secure authentication protocol and is invulnerable to DoS attacks. In addition, an experimental validation of the client-puzzle problem determined that the challenge technique can be adjusted to implement different difficulty levels of the puzzle (Chapter 5).

3. **Investigate, develop, and validate a protocol suite to defend against internal DoS attacks.**

- The risks of internal cloud-based DoS attacks were investigated. Furthermore, an existing authorization protocol in cloud computing was examined (Chapter 2).

- A second module of the cloud-based secure authentication protocol (CSAM-2) that defends against internal DoS attacks was proposed. This protocol can be implemented on public and hybrid deployment models of the cloud system. Multiple techniques were integrated into the design of the CSAM-2 protocols, such as the client puzzle problem, extended unique encryption text (EUET) application, and deadlock avoidance algorithms, to prevent security flaws that may incur DoS attacks. The design of the proposed CSAM-2 protocol implements a deadlock avoidance algorithm to allocate the authenticated cloud user to his/her requested service with avoidance of risks of internal cloud-based DoS attacks. Additionally, in the CSAM-2 protocol suite design, the host of the requested service authenticates the cloud user in a lightweight authentication process. The CSAM-2 protocol suite was developed based on basic hardware and software requirements for protocol participants. Thus, the protocol is practically applicable in multiple cloud service models, specifically, SaaS, PaaS, and IaaS (Chapter 4).

- The experimental validation of the CSAM-2 protocol suite indicated that the CSAM-2 protocol is a secure authentication protocol that is lightweight, reliable, and scalable. The reliability of the protocol is based on the fact that the deadlock avoidance algorithm was implemented without cache processing. In addition, the cloud resources are not overloaded when performing high computation processes of the protocol. The scalability of the protocol is attributed to the fact that the maximum accessibility number of each service is adjustable based on the cloud service properties. Thus, the protocol is expandable with the size of the cloud system without affecting the cloud resources (Chapter 5).

6.2 Future Work

This study examines the ability of a system to detect DoS attacks during authentication processes through different cloud-computing service models. The proposed future work will address the following key aspects of this research topic.

- The proposed work in this research considers using the service allocation technique to allocate requested services to users while preventing the risk of internal cloud-based DoS attacks. However, this research does not consider the implementation of existing resource-allocation techniques, such as allocating services to a specific or the nearest data center in the cloud computing system. Future work should consider the integration of the resource-allocation techniques with existing service allocation technique in this research.

- In the design of authentication and identification protocol, the time threshold between multiple consecutive attempts to prevent further requests from the same user is an open research area. Further statistical research is needed to determine the acceptable time threshold.

- The present research does not consider existing implementations of software or hardware firewalls or intrusion detection systems (IDS). Conducting such research with existing firewalls and IDSs remains an open research area that future work should address.

References

Arora, K., Kumar, K., & Sachdeva, M. (2011). Impact analysis of recent DDoS attacks. *International Journal on Computer Science & Engineering*, *3*(2), 877–884.

Boyd, R. (2012). *Getting Started with OAuth 2.0*. Sebastopol, CA: O'Reilly Media.

Burrows, M., Abadi, M., & Needham, R. M. (1989). A Logic of Authentication. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, *426*(1871), 233–271. http://doi.org/10.1098/rspa.1989.0125

Choudhury, A. J., Kumar, P., Sain, M., Lim, H., & Jae-Lee, H. (2011). A Strong User Authentication Framework for Cloud Computing. In *IEEE Asia-Pacific Services Computing Conference* (pp. 110–115). IEEE. http://doi.org/10.1109/APSCC.2011.14

Darwell, B. (2013). Study: percent of consumers logging into sites with Facebook dips as Google gains. Retrieved August 27, 2015, from http://www.adweek.com/socialtimes/study-percent-of-consumers-logging-into-sites-with-facebook-dips-as-google-gains/292224

Dierks, T. (1999). The TLS Protocol Version 1.0. Retrieved from http://tools.ietf.org/html/rfc2246

Diffie, W., & Hellman, M. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644–654. http://doi.org/10.1109/TIT.1976.1055638

Dijkstra, E. W. (2002). *Cooperating sequential processes*. Springer.

Dittrich, D., Mirkovic, J., Reiher, P., & Dietrich, S. (2004). *Internet Denial of Service: Attack and Defense Mechanisms*. Pearson Education.

Fall, K., & Varadhan, K. (2007). *The network simulator (ns-2)*. Retrieved from http://www.isi.edu/nsnam/ns

Fu, D., & Shi, F. (2012). Buffer Overflow Exploit and Defensive Techniques. *2012 Fourth International Conference on Multimedia Information Networking and Security*, 87–90. http://doi.org/10.1109/MINES.2012.81

Gonzalez, J. M., Anwar, M., & Joshi, J. B. D. (2011). A trust-based approach against IP-spoofing attacks. *2011 Ninth Annual International Conference on Privacy, Security and Trust*, 63–70. http://doi.org/10.1109/PST.2011.5971965

Google Developers. (2014). Using OAuth 2.0 for Web Server Applications - Google Accounts Authentication and Authorization. Retrieved August 27, 2015, from https://developers.google.com/identity/protocols/OAuth2?hl=de

Hammer-Lahav, E. (2010). The OAuth 1.0 Protocol. *Internet Engineering Task Force IETF*. Retrieved from http://tools.ietf.org/html/rfc5849

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. *No. RFC 6749*, 85. http://doi.org/10.1109/MIC.2012.11

Hwang, M. S., Chong, S. K., & Chen, T. Y. (2010). DoS-resistant ID-based password authentication scheme using smart cards. *Journal of Systems and Software*, *83*(1), 163–172. http://doi.org/10.1016/j.jss.2009.07.050

Jaidhar C. D. (2012). Enhanced Mutual Authentication Scheme for Cloud Architecture. In *3rd IEEE International Advance Computing Conference (IACC)* (pp. 70–75). IEEE.

Juels, A., & Brainard, J. (1999). Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Network and Distributed System Security Symposium (NDSS)* (pp. 151–165). Internet Society.

Kim, M., Fujioka, A., & Ustaoğlu, B. (2009). Strongly Secure Authenticated Key Exchange without NAXOS' Approach. In T. Takagi & M. Mambo (Eds.), *Advances in Information and Computer Security* (Vol. 5824, pp. 174–191). Berlin: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-04846-3_12

Kliazovich, D., Bouvry, P., & Khan, S. U. (2012). GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. *Journal of Supercomputing*, *62*(3), 1263–1283. http://doi.org/10.1007/s11227-010-0504-1

Lemon, J. (2002). Resisting SYN flood DoS attacks with a SYN cache. In *Proceedings of the BSD Conference 2002 on BSD Conference* (p. 10). Berkeley, CA, USA: USENIX Association.

Lenstra, A. K., Lenstra Jr., H. W., & Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, *261*(4), 515–534. http://doi.org/10.1007/BF01457454

LoginRadius. (n.d.). EasyToBook: Simple Registration and Boosted User Engagement. Retrieved August 27, 2015, from http://www.loginradius.com/easytobook-simplify-registration-and-boost-user-engagement

Manber, U. (1989). *Introduction to Algorithms: A Creative Approach* (1st ed.). Addison-Wesley.

Mather, T., Kumaraswamy, S., & Latif, S. (2009). *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. Sebastopol, CA: O'Reilly Media.

Meadows, C. (2001). A Cost-Based Framework for Analysis of Denial of Service in Networks. *Journal of Computer Security*, *9*(1-2), 143–164.

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. *National Institute of Standards and Technology, Information Technology Laboratory* (Vol. 145). Retrieved from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

Moskowitz, R., Nikander, P., Jokela, P., & Henderson, T. (2008). Host Identity Protocol. Retrieved April 5, 2015, from http://tools.ietf.org/html/rfc5201

Nazario, J. (2008). DDoS attack evolution. *Network Security*, *2008*(7), 7–10. http://doi.org/10.1016/S1353-4858(08)70086-2

Neumann, P. G. (2000). Inside Risks: Denial-of-Service Attacks. *Communications of the ACM*, *43*(4), 136. http://doi.org/10.1145/332051.332797

Nielsen, J. (1993). *Usability engineering* (1st ed.). San Francisco: Morgan Kaufmann.

Pacini, E., Ribero, M., Mateos, C., Mirasso, A., & Garino, C. G. (2012). Simulation on cloud computing infrastructures of parametric studies of nonlinear solids problems. In F. Cipolla-Ficarra, K. Veltman, D. Verber, M. Cipolla-Ficarra, & F. Kammüller (Eds.), *Advances in New Technologies, Interactive Interfaces and Communicability* (Vol. 7547, pp. 58–70). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-34010-9_6

Parallels. (2010). *Delivering Extraordinary Density for Cloud Service Providers*.

Patel, C. M., & Borisagar, V. H. (2012). Survey On Taxonomy Of DDoS Attacks With Impact And Mitigation Techniques. *International Journal of Engineering Research & Technology (IJERT)*, *1*(9), 1–8.

Salomaa, A. (1996). *Public-Key Cryptography* (2nd ed.). Berlin: Springer.

Schuba, C. L., Krsul, I. V., Kuhn, M. G., Spafford, E. H., Sundaram, A., & Zamboni, D. (1997). Analysis of a denial of service attack on TCP. *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, 208–223. http://doi.org/10.1109/SECPRI.1997.601338

Siriwardena, P. (2014). *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE*. New York, NY: Apress.

Southern, E., Ouda, A., & Shami, A. (2013). Wireless security: securing mobile UMTS communications from interoperation of GSM. *Security and Communication Networks*, *6*(4), 498–508. http://doi.org/10.1002/sec.674

Syverson, P., & Cervesato, I. (2001). The Logic of Authentication Protocols. In *Foundations of Security Analysis and Design* (Vol. 2171, pp. 63–137). Springer. http://doi.org/10.1007/3-540-45608-2_2

Syverson, P. F., & Van Oorschot, P. C. (1994). On Unifying Some Cryptographic Protocol Logics. In *IEEE Computer Society Symposium on Research in Security and Privacy* (pp. 14–28). http://doi.org/10.1109/RISP.1994.296595

Tritilanunt, S., Boyd, C., Foo, E., & Nieto, J. M. G. (2007). Toward Non-parallelizable Client Puzzles. In F. Bao, S. Ling, T. Okamoto, H. Wang, & C. Xing (Eds.), *Cryptology and Network Security* (Vol. 4856, pp. 247–264). Berlin: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-76969-9

Tsaur, W. J., Li, J. H., & Lee, W. Bin. (2012). An efficient and secure multi-server authentication scheme with key agreement. *Journal of Systems and Software*, *85*(4), 876–882. http://doi.org/http://dx.doi.org/10.1016/j.jss.2011.10.049

Twitter. (2015). Tokens from dev.twitter.com. Retrieved August 27, 2015, from https://dev.twitter.com/oauth/overview/application-owner-access-tokens

Wang, H., Jin, C., & Shin, K. G. (2007). Defense Against Spoofed IP Traffic Using Hop-Count Filtering. *IEEE/ACM Transactions on Networking*, *15*(1), 40–53. http://doi.org/10.1109/TNET.2006.890133

Yahoo. (2015). Yahoo OAuth 2.0 Guide. Retrieved August 27, 2015, from https://developer.yahoo.com/oauth2/guide/index.html

Yassin, A. A., Jin, H., Ibrahim, A., Qiang, W., & Zou, D. (2013). Cloud Authentication Based on Anonymous One-Time Password. In Y.-H. Han, D.-S. Park, W. Jia, & S.-

S. Yeo (Eds.), *Ubiquitous Information Technologies and Applications* (Vol. 214, pp. 423–431). New York: Springer Dordrecht Heidelberg. http://doi.org/10.1007/978-94-007-5857-5_46

Yassin, A. a., Jin, H., Ibrahim, A., & Zou, D. (2012). Anonymous Password Authentication Scheme by Using Digital Signature and Fingerprint in Cloud Computing. In *Second International Conference on Cloud and Green Computing* (pp. 282–289). IEEE. http://doi.org/10.1109/CGC.2012.91

Appendices

Appendix A: The results of the simulation using GreenCloud simulator tool

Table A.1. VM load with different processes

Number of requests/second	VM load				
	Token storage	AES encryption	AES decryption	SHA-512 hashing	Generating random number
300	0.1875	0.2623	0.2561	0.2439	0.2373
310	0.2170	0.2676	0.2584	0.2527	0.2462
320	0.2055	0.2772	0.2703	0.2596	0.2508
330	0.2216	0.2914	0.2780	0.2688	0.2561
340	0.2266	0.3006	0.2906	0.2742	0.2788
350	0.2324	0.3106	0.2941	0.2784	0.2753
360	0.2389	0.3133	0.3029	0.2872	0.2826
370	0.2458	0.3198	0.3083	0.2995	0.2933
380	0.2485	0.3332	0.3175	0.3033	0.3021
390	0.2596	0.3428	0.3271	0.3094	0.2964
400	0.2650	0.3574	0.3367	0.3221	0.3206
410	0.2699	0.3639	0.3474	0.3301	0.3259
420	0.2753	0.3708	0.3531	0.3382	0.3336
430	0.2799	0.3823	0.3631	0.3470	0.3367
440	0.2857	0.3861	0.3692	0.3524	0.3478
450	0.3018	0.3992	0.3811	0.3650	0.3482
460	0.2991	0.4068	0.3923	0.3823	0.3639
470	0.3110	0.4137	0.3957	0.3808	0.3765
480	0.3106	0.4256	0.4045	0.3888	0.3804
490	0.3248	0.4340	0.4145	0.3961	0.3846
500	0.3309	0.4406	0.4233	0.4018	0.3942

Table A.2. VM memory load with different processes

Number of requests/second	VM memory load				
	Token storage	AES encryption	AES decryption	SHA-512 hashing	Generating random number
300	0.0002	0.2499	0.2385	0.1931	0.1226
310	0.0002	0.2595	0.2567	0.2393	0.2162
320	0.0002	0.2660	0.2676	0.2622	0.2537
330	0.0002	0.2707	0.2753	0.2754	0.2727
340	0.0081	0.2757	0.2803	0.2849	0.2842
350	0.0553	0.2772	0.2851	0.2904	0.2930
360	0.1719	0.2791	0.2881	0.2953	0.2980
370	0.2573	0.2802	0.2898	0.3002	0.3029
380	0.2973	0.2819	0.2913	0.3018	0.3075
390	0.3187	0.2835	0.2928	0.3037	0.3088
400	0.3334	0.2852	0.2944	0.3048	0.3109
410	0.3431	0.2872	0.2962	0.3065	0.3120
420	0.3507	0.2882	0.2979	0.3082	0.3135
430	0.3557	0.2896	0.2998	0.3098	0.3152
440	0.3606	0.2926	0.3006	0.3115	0.3169
450	0.3658	0.2962	0.3024	0.3134	0.3186
460	0.3677	0.3019	0.3055	0.3143	0.3205
470	0.3693	0.3081	0.3092	0.3161	0.3214
480	0.3712	0.3168	0.3148	0.3192	0.3231
490	0.3722	0.3259	0.3208	0.3229	0.3261
500	0.3739	0.3351	0.3298	0.3286	0.3298

Table A.3. VM storage load with different processes

Number of requests/second	VM storage load				
	Token storage	AES encryption	AES decryption	SHA-512 hashing	Generating random number
300	0.0004	4.797E-07	4.580E-07	7.722E-08	4.904E-08
310	0.0004	4.983E-07	4.928E-07	9.571E-08	8.648E-08
320	0.0004	5.106E-07	5.138E-07	1.049E-07	1.015E-07
330	0.0004	5.198E-07	5.285E-07	1.102E-07	1.091E-07
340	0.0155	5.293E-07	5.382E-07	1.140E-07	1.137E-07
350	0.1062	5.321E-07	5.474E-07	1.162E-07	1.172E-07
360	0.3300	5.359E-07	5.532E-07	1.181E-07	1.192E-07
370	0.4939	5.379E-07	5.564E-07	1.201E-07	1.212E-07
380	0.5709	5.412E-07	5.592E-07	1.207E-07	1.230E-07
390	0.6118	5.443E-07	5.621E-07	1.215E-07	1.235E-07
400	0.6400	5.476E-07	5.653E-07	1.219E-07	1.244E-07
410	0.6587	5.514E-07	5.687E-07	1.226E-07	1.248E-07
420	0.6734	5.534E-07	5.719E-07	1.233E-07	1.254E-07
430	0.6830	5.561E-07	5.756E-07	1.239E-07	1.261E-07
440	0.6923	5.619E-07	5.771E-07	1.246E-07	1.267E-07
450	0.7023	5.688E-07	5.805E-07	1.254E-07	1.275E-07
460	0.7060	5.797E-07	5.865E-07	1.257E-07	1.282E-07
470	0.7090	5.915E-07	5.936E-07	1.264E-07	1.286E-07
480	0.7127	6.083E-07	6.043E-07	1.277E-07	1.292E-07
490	0.7146	6.257E-07	6.160E-07	1.292E-07	1.304E-07
500	0.7178	6.435E-07	6.332E-07	1.314E-07	1.319E-07

Curriculum Vitae

Name: Marwan Darwish

Post-secondary Education and Degrees:

King Abdulaziz University
Jeddah, Saudi Arabia
1998-2003 B.Sc. Electrical and Computer Engineering

King Abdulaziz University
Jeddah, Saudi Arabia
2005-2008 M.Sc. Electrical and Computer Engineering

The University of Western Ontario
London, Ontario, Canada
2011-2015 Ph.D. Electrical and Computer Engineering

Honours and Awards:

King Abdulziz University Scholarship
2010-2015

Related Work Experience:

Web Developer
King Abdulaziz University – Information Technology Deanship
2003-2004

Technical Support Manager of e-government project
Integrated Visions Company (Saudi Arabia)
2004-2005

Instructor & Head of Computer Technology Department
Technical and Vocational Training Corporation
2005-2009

Lecturer
King Abdulaziz University – Jeddah Community College (JCC)
2009-2010

Teaching Assistant
The University of Western Ontario
2011-2015

Publications:

Darwish M, Ouda A, Capretz LF. Cloud-based DDoS Attacks and Defenses. International Conference on Information Society (i-Society), Toronto, Canada, June 2013. p. 67–71.

Darwish M, Ouda A, Capretz LF. Formal Analysis of an Authentication Protocol Against External Cloud-Based Denial-of-Service (DoS) Attack. International Journal for Information Security Research (IJISR). 2013;3(1/2):400–7.

Darwish M, Ouda A, Capretz LF. A cloud-based secure authentication (CSA) protocol suite for defense against Denial of Service (DoS) attacks. Journal of Information Security and Applications. 2015 Jan;20:90–8.

Darwish M, Ouda A. Evaluation of an OAuth 2.0 Protocol Implementation for Web Server Applications. 6th International Conference and Workshop on Computing and Communication, Vancouver, Canada, October 2015.

Darwish M, Ouda A. An Enhanced Cloud-Based Secure Authentication (ECSA) Protocol Suite for Prevention of Denial-of-Service (DoS) Attacks. Future Generation Computer Systems. 2015; (Submitted).