USER CENTERED SECURITY SERVICE LEVEL AGREEMENT ENFORCEMENT MECHANISMS

by

Sultan Alasmari

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Software and Information Systems

Charlotte

2022

Approved by:

Dr. Weichao Wang

Dr. Jinpeng Wei

Dr. Mohamed Shehab

Dr. Jun-tao Guo

ABSTRACT

SULTAN ALASMARI. User Centered Security Service Level Agreement Enforcement Mechanisms. (Under the direction of DR. WEICHAO WANG)

As businesses profoundly rely on cloud services security becomes a critical concern. Various emerging technologies depend on cloud computing for its intrinsic features such as scalability, storage and cost-effectiveness. While this may be true, cloud users are wary about the confidentiality and integrity of outsourced data. Security Service Level Agreement(SSLA) provides transparency between service providers and customers to guarantee security services terms are delivered as agreed in the SSLA. Since many corporations outsource security services to cloud providers, it appears necessary to develop a user-centered SSLA enforcement mechanism to verify service provider commitment.

In this dissertation, the main objective is to design and adopt user centered service level agreement security enforcement mechanisms to verify the execution of SSLA and hence detect SSLA violations. First, we tackle the problem of Proof Of Encryption (PoE) and then propose two security mechanisms to verify the encryption operation by the service provider whether both parties have the encryption keys or only the service provider maintains the key. Second, we developed a security enforcement mechanism where the service requester chooses one service provider to negotiate Partial Homomorphic Encryption (PHE) algorithm so that the service requester can only query encryption results at the service provider without disclosing the ciphertext. Another SSLA security enforcement mechanism is proposed to verify third-party network scanning. A customer can verify if SSLA is violated or not by relying on a group of tester nodes called *bots* to do the testing. Finally, in order to engage more nodes to participate in the network scanning verification, one future direction would be to develop an incentive model to motivate nodes who launch the network scanning

to maximize their profit and attract more nodes. The results show that our security approaches are able to detect a deceptive service provider with high probability while reducing the overhead on the service requester which paves the way to design effective SSLA enforcement mechanisms.

ACKNOWLEDGEMENTS

I am grateful for my dissertation committee chair Professor Weichao Wang for his guidance, encouragement and patience in completing this dissertation. I am also thankful for Professor Yu Wang for his advice and mentoring during the journey.I would like to extend my thanks to the dissertation committee members for their valuable insight including Dr.Mohammed Shehab, Dr. Jinpeng Wei and Dr. Jun-tao Guo.

I would like to thank my wife Daryen and my wonderful kids Omar and Mila for being supportive during the PHD journey. I would like to thank my family especially my father Ali who taught me perseverance and hard work and my lovely mom Alwah who inspired me with kindness and wisdom and all my siblings for their loving and support during the journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CTP Cloud Trust Protocol.

HE Homomorphic Encryption.

PHE Partially Homomorphic Encryption.

PoE Proof of Encryption.

SLO Service Level Objective.

SSLA Security Service Level Agreement.

TTPP Trusted Third Party Provider.

CHAPTER 1: Introduction

This chapter is organized as follows: Section 1.1 introduces the research motivation, then Section 1.2 describes the concept of Security Service Level Agreement (SSLA) in the cloud model. Then, Section 1.3 discusses the chapters contribution and Section 1.4 outlines the research proposal.

1.1 Motivation

Recently, the cloud-edge computing paradigm has attracted great attention from both academia and corporations for its great value in reducing costs, providing better utilization of resources and ensuring the continuity of services provided to customers. Furthermore, the cloud-edge services provide stability and easy use, and execute massive amounts of computation. Gartner research group predicts that public cloud providers will earn more than 100 billion dollars by 2020, only by providing Cloud Application Services (SaaS) to customers [Gartner(2019)]. Small and Medium Enterprises (SMEs) rely heavily on cloud providers to reduce operation costs and receive reliable services. The main common cloud providers can be categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) ,Software as a Service (SaaS) and Security as a Service (SECaaS). The growing popularity of cloud computing providers has directly stimulated businesses and investors to outsource services by providing affordable security services. Conversely, the lack of security enforcement mechanisms and the massive number of incidents continue to occur causes users to be wary about the adoption of cloud services.

As a result, security researchers have argued that storing files in the cloud could introduce new threats. For instance, a recent report published by Symantec [Syman-

tic(2019)] indicated alarming results where one of the top threats is managing identity and authentication environments, where 65% of users do not even apply multi-factor authentication and 70% percent do not use encryption. In many cases, cloud security compliance mostly relies on third parties to check whether the provider is compliant with security standards and best practices. Another way to verify security property compliance is by applying attestation and cryptographic techniques to verify cloud compliance from the users side. This is cost effective and ensures the user cannot rely on third party providers.

Cloud providers and customers sign a Security Service Level Agreement (SSLA), which is an agreement between the service requester and cloud-edge provider to identify cloud security services, metrics agreed upon and overall the responsibilities of the service provider. This agreement entails that the providers are required to sign and adhere to Security Service Level Agreements (SSLAs) to set specific performance and security metrics. These metrics are the quantifiable measurements that are mandated in the agreement to notify the user if any violation occurs. In cloud-edge computing, the customer is incapable of quantifying the cloud-edge services the user is receiving. Thus, cloud providers guarantee the protection of customer stored data and applications running by either outsourcing security services to third parties or allowing the user to apply security controls, such as encrypting one's own data and using sanitization methods to protect that data.

Cloud computing has been facing several security threats that impact cloud customers severely, such as data breaches, insufficient-access control management, and limited cloud usage visibility [Alliance(2019)]. First, one of the major threats is data breaches, where confidential information is disclosed by a malicious user or entity. The lack of cloud user knowledge of the undetected threat, if it is detected, exacerbates the problem and causes uncertainty about whether the data is being protected from the cloud service provider or any third party involved with the service provider.

Secondly, weak identity and access management policies, the lack of protecting credentials, cryptographic keys, and digital certificates expose cloud resources to serious threats. Finally, this breach occurs when the company is unable to detect whether the cloud service model that is being adopted is secure or malicious. Consequently, the occurrence of these significant threats causes system security researchers to design audit schemes and security approaches to investigate the credibility of cloud-edge providers.

This research domain has several challenges, such as the lack of disclosing security enforcement metrics from cloud providers due to maintaining the confidentiality and non-disclosure agreement with customers. Disclosing sensitive information about specific security controls being adopted would allow the attackers to infiltrate through the network or system using the announced published security parameters.

The second challenge is developing a lightweight security enforcement mechanism from the user's side to verify data security,but, this is not a straightforward problem. The data security verification is intricate because of the complex security issues of the cloud model and a few cryptography based security mechanisms exist that assume users can audit cloud providers without their knowledge. One of the main papers in this domain proposed Proof of Retrievability (POR) [Shacham and Waters(2008)], which supports public auditing schemes. Next, this dissertation is precisely interested in studying the problem of file encryption and network scanning in cloud outsourcing, which has never been discussed in the literature.

Despite the enormous benefits the cloud services provide, the service provider cheats the user by degrading security services to reduce the computation cost by violating the SSLA, while storing the data in the cloud. Therefore, several papers have focused on developing verification schemes to detect if the provider is cheating or trying to reduce the SLO or metrics value parameter to gain a competitive advantage over other service providers. Several approaches have been proposed to study the remote verification of

security properties to ensure that the confidentiality and integrity of the data stored in the cloud are honored as agreed in the SSLA. First, the researchers investigated the integrity of the outsourced data in the cloud to discover whether the stored file is corrupted due to internal malicious use instance [Shacham and Waters(2008)].

Secondly, the thread of research is focused on developing audit schemes for cloud outsourced data to guarantee data sharing in the cloud environment, while hiding sensitive information [Wang et al.(2010)Wang, Wang, Ren, and Lou]. It is noticeable that most of the provided security approaches assume that there is a third-party auditor (TPA) residing between the cloud and the user side to conduct the verification and enforcement of security properties. In addition, the approaches are focused mainly on guaranteeing the integrity of the data by ensuring that the modification of the file blocks are maintained and tracked. Nevertheless, the trusted third party can be compromised, thus, disclosing confidential client information. Consequently, user-centric enforcement mechanisms are needed to conduct verification directly from users to cloud-edge providers with no other third-party intervention. However, in this research approach, this proposal assumes that the cloud provider who outsources customer's data is not to be trusted. Therefore, there is a need to develop a novel security enforcement mechanism to verify file encryption in the cloud. End users of cloud-edge computing services may have an agreement with cloud providers regarding file encryption algorithms, verifying the key length, and maintaining this level of protection. For instance, the agreement may specifically state that the cloud-edge provider shall encrypt the data with AES algorithm and the key size 256-bit length. However, how the end users are able to verify that the level of encryption is maintained. Thus, one way to achieve this is by selecting a random chunk of data and sending several challenges to the provider without knowing what data segment has been selected. The cloud-edge provider will not be able to identify the selected data for verification. The user adapts a challenge-and-response approach to test whether

the provider is going to maintain the same level of protection on the outsourced data without knowing that they are being verified from the user side. For such approaches, maintaining low overhead while verifying the data is required to guarantee the efficiency of the proposed approach. Lastly, ensuring that the provider is not capable of applying encryption on the fly attacks, which has been detected in our approach, is needed to detect SSLA violations. In the context of cloud computing, there are two common security audit approaches. Public auditing is where the user relies on a third-party auditor to handle verification sent to the cloud provider. Private auditing is directly conducted between the user and the cloud provider. In this research, we use private auditing techniques to verify several SSLA metrics are honored by the service provider.

1.2 Problem Statement

Cloud audits are security mechanisms to verify and validate cloud provider's commitment to the service requester to make sure the cloud provider is genuine and trustworthy. We aim to develop several user based security mechanisms to detect SSLA violations.

1.3 Contribution

This dissertation presents multiple security mechanisms to enforce security service level agreement (SSLA) properties to detect SSLA violations.First we discuss the Proof of Encryption (PoE), we define the expected properties of PoE mechanisms,then we design two mechanisms for PoE that allow end users to verify the encryption operations by service provider. Next, we study the problem of proof of outsourced encryption, we begin with defining the expected properties and propose the symmetric and homomorphic encryption cases and design the challenge and verification procedures. We propose an infrastructure through which an end user can test its network security services with configurable frequency and self-crafted test cases.

The integrity and authenticity of the test are properly protected. Third,we analyze the detection capability and overhead of the proposed approach. In the fourth contribution, an incentive model is developed to incentivise network nodes to participate in the network security verification.

1.4 Outline

The rest of this dissertation is organized as follows: We discuss the SSLA Cloud major concepts in Chapter2. Then, we review some related works in chapter 3. In Chapter 4 we propose the proof of encryption, the detailed description of the security approach is presented. In Chapter 5 we present the proposed approach for the Proof of Outsourced Encryption in detail and analyze the correctness of the proposed protocol using BAN logic. Next, we present the Proof of Network Security Scanning Services and discuss the results in Chapter 6. In Chapter 7 we present the last contribution Incentivisation of outsourced network testing. Last but not least we conclude our work and discuss the future work in Chapter 8.

CHAPTER 2: Security Service Level Agreement

2.1 SSLA Definition

The National Institute of Standards and Technology (NIST) [NIST2(2015)] defines Service Level Agreement as "A service contract between a service provider and a service-using organization that defines the level of service to be provided,such as the time to recover from an operational failure or a system compromise". This legal agreement between several parties entails terms of services where a provider offers security services or leases security appliances to customers. The SSLA also touches upon data confidentiality, rights to access the data and states each party restrictions and penalties when a violation is detected. In order to measure Service Provider's commitment with the SSLA,the agreement communicates Service Level Objective (SLO) and the value assigned for each objective.

2.2 SSLA Benefits

The SSLA protects users and provider's rights by guaranteeing transparent and dependable services. The following are some key benefits the SSLA would provide to satisfy all parties and manage conflicts in case of violations:

- Clearness: The SSLA document should be written in a clear way, so it can be understood and explained to the parties involved.
- Doable : The services and metrics shall be achieved by the provider as committed or a compensation shall be paid to users.
- Enforceable: The SSLA security service metrics can be enforced and validated by users and third party auditors.

- Quantifiable: The SSLA security services can be measured to evaluate the effectiveness of the security services while running.

2.3 SSLA key entities

The SSLA involves key entities that are engaged not only in the SSLA planning , negotiation but also in verification and enforcement :

1. **Users** : Cloud users are the data owners, they are capable of uploading and downloading files and running web services on the cloud instead of hosting physical servers and network devices to run the business with less cost.

2. **Service Providers** : The provider runs a massive amount of networking and infrastructure appliances to gain profit and provide cost effective services for customers.

3. **Third Party Auditor** : This entity works with the user to verify that service provider services are compliant with SSLA terms and metrics.

2.4 SSLA Elements

Service Description: The Service Provider publishes the service description to outline cloud services, the charged metrics such as MTTD (meantime to detect) as specified in the agreement. Moreover, the service description specifies functions and the time period when the users are charged either monthly or based on the usage of the metrics.

Service Credit: The amount of money the Service Provider credits cloud users when a violation occurred. Once a violation is detected the service provider shall apply the SSLA renegotiation phase to establish or terminate the agreement and compensate the cloud user. It is important to note that the two parties might not reach an agreement that could lead to a service termination.

Security Metrics: As mentioned in the previous section that the security metrics are assigned a value to measure SSLA service commitment. The value is represented as a percentage or a Boolean condition.These metrics vary from one cloud security service to another based on the frequency the value should be measured. The metrics are classified into two categories,technical metrics, to quantify the effectiveness of the cloud service and to enforce certain security controls for monitoring. The second type of the security metrics are the managerial metrics, this category includes the security policies, standards and procedures that the service provider is bound by.

Payments and Exclusions: The cloud users shall request a credit when the Service Provider commitment is not met or a violation is detected. The SSLA exclusions express the factors that are out of control and the provider is not responsible for either due to user negligence or an interference of a third party entity.

2.5 SSLA cloud Standards

Cloud computing is widely adapted, the adherence to well-established standards are becoming crucial to regulate all cloud services. The following are some standards accompanied the cloud model:

- US Health Insurance Portability Accountability Act (HIPPA) [HIPPA(2015)]
- US Sarbanes-Oxley Act(SOX) [SAOX(2022)]
- Payment Card Industry Data Security Standard(PCI DSS) [PCI-DSS(2022)]
- Open Web Application Security Project(OWASP) [OWASP2(2022)]
- US Fedral Information Security Management Act (FISMA) [FISMA(2022)]

The cloud users play a big role in the cloud audit and enforcement, several approaches developed to meet users needs and allow them to gain faith in service providers. The models provide a baseline foundation for cloud technical and administrative controls :

Cloud STAR Roadmap: This model is developed by Cloud Security Alliance (CSA) , the Security Trust Assurance and Risk(STAR) [Standard(2022)] is a very solid framework to audit cloud resources and ensure comparability among standards.

Cloud Controls Matrix: is a framework consisting of cloud security controls, the standard provides very detailed and structured security controls that cover several security domains.Some experts claim that the standard is the currently effective standard.

C5 Cloud Controls [Control(2022)]: The cloud computing compliance catalog is developed by the German Federal Office for information security, it combines several standards such as ISO 27001, cloud security matrix and BSI publications.

2.6 Threat Model

In this work, we assume that the Service Provider is untrusted, the provider does not intentionally jeopardize data confidentiality and integrity to disclose user information. However, the provider has the intention to downgrade or evade services as agreed in the SSLA to serve his performance needs. Given the above assumption, we aim to design several SSLA enforcement mechanisms to detect a service provider who violate the SSLA.

2.7 SSLA Cloud Architecture

Generally, the cloud model is organized into two categories:

1. Single_tenant: This type of architecture is designed for businesses who prefer to run a separate infrastructure, it is usually customized for the customer to provide better security for data protection.

2. Multi_tenant: The cloud server is basically shared among several cloud customers where all data partitions has the same hardware and platform configurations. Therefore, several entities have access to the cloud resource and thus the risk of storing confidential data and running web services may lead to data

theft and disclosure.

Having established the differences between single tenant and multi-tenant cloud, it is very important to mention that the cloud has several deployment models such as public, private and hybrid cloud. The private cloud allows more controls on the assets, while the public cloud controls all software and hardware and offers only services to users.

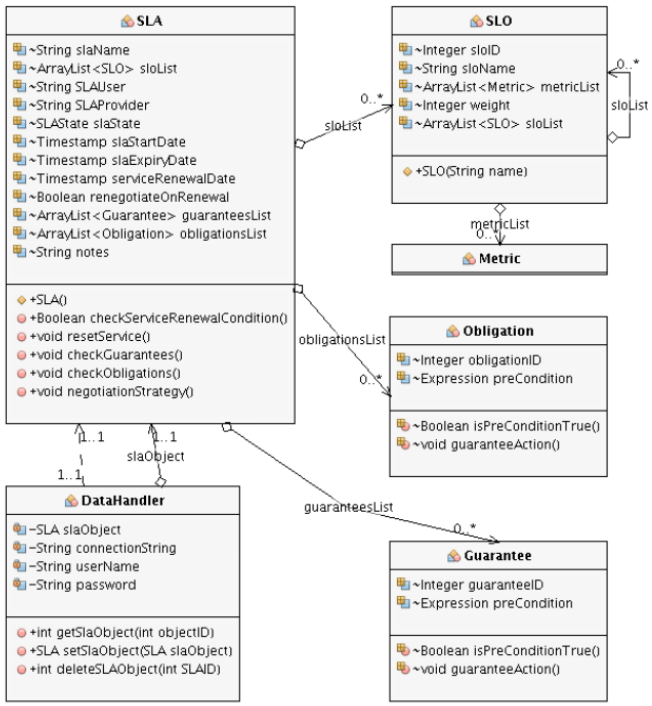


Figure 2.1: SLA Structure diagram

The security SLA comprises several phases including the negotiation, agreement signing, implementation, enforcement and continuous monitoring for verification, and finally remediation to compensate customers. Several cloud Security SLA frameworks are proposed such as SPECS [Casola et al.(2015)Casola, De Benedictis, Rak, and Villano] and SLA Security by Design (SSDE) [Casola et al.(2020b)Casola, De Benedictis, Rak, and Villano]. SPECS framework is a European project aims to automate the SSLA life cycle starting from negotiation to enforcement in PaaS (Platform as a Service). The SPECS application integrated with the cloud services to administer

security SLA stages while connecting all cloud services modules. Furthermore, one key feature of SPECS is enabling agents to monitor SLO parameters, then generate and collect data that are refined by the enforcement agent to detect an SSLA violation. The SSDE model is developed to facilitate the development of a secure cloud application to tackle security risks and combine security in the overall development process. The idea is to identify software vulnerability while conducting security assessment, the vulnerability is divided into categories such as data ownership or data integration to mitigate vulnerabilities earlier in the design process.

Another Security SLA framework called LoM2HiS is developed [Emeakaroha et al.(2010)Emeakaroha, Brandic, Maurer, and Dustdar] to assign a security metric to specific high level security parameters. The enforcement module does the metric assignment to SLO for run time monitoring. The ultimate goal is to monitor security metrics and automate the security enforcement to detect SSLA violations. Specifying Security requirements studied by researchers [Kaaniche et al.(2017)Kaaniche, Mohamed, Laurent, and Ludwig] , they designed an SSLA monitoring system and a description language relying on rSLA model and their enhanced framework Sec-rSLA aims to enforce SSLA by dividing the cloud service into elements and then assign a measured security properties for each element.

CHAPTER 3: Related Work

In this chapter we present and discuss prior research that relates to the problem of $SSLA$ knowledge representation and verification for outsourced data. We also explore previous research that investigates user based security verification and enforcement approaches for data.

3.1 SSLA Knowledge Representation

The notion of knowledge representation is concerned with expressing description of an object to reason about the entities and their relationships. During SSLA provisioning phase, SSLA model representation is inarguably mandatory to devise SSLA contracts, parties involved and processes in place for compliance and verification of SSLA contract. The way in which SSLA is represented was studied extensively, in [Hale and Gamble(2013)], [Lee et al.(2015)Lee, Kavi, Paul, and Gomathisankaran] several SSLA cloud compliance terms ontology were developed and then each SSLA term is linked to a security control relying on a security standard and federal regulation such as NIST SP800-53 and HIPAA. The purpose is to automate the process of SSLA provisioning where each security concept is represented as a class with reference to standards. Furthermore, SSLA contract establishment ontology is presented [Griffo et al.(2019)Griffo, Almeida, Guizzardi, and Nardi] to describe SSLA contract overall life cycle and describe each customer and a service provider legal involvement in the SLA contract to specify their roles and responsibilities. Afterward, the devised ontology is used as an input to Archimate language model to express SSLA service contract entities. However, this approach is limited to external entities and it overlooks individuals involvement inside an organization in SSLA compliance.

3.2 Cloud Audit

The major cloud audit models can be categorized into public audit and private audit approaches. The public audit ensures that the Trusted Third Party Provider (TTPP) is involved while conducting the security audit. On the other hand, the private audit enables users to design their own verification mechanism independently [Zhou et al.(2018)Zhou, Fu, Yu, Su, and Kuang]. Previous lines of research were interested in studying public audit schemes, the authors [Kim and Jeong(2017)] proposed a public auditing scheme for cloud shared data using constant verification time. Their scheme is resilient against key replacement attack and passive KGC attack. Tian *et* all [Tian et al.(2015)Tian, Chen, Chang, Jiang, Huang, Chen, and Liu] developed a public batch auditing scheme where the TPA uses Dynamic Hash Table (DHT) to keep track of the audit task and minimize computation and communication overhead.

There is undoubtedly a need for a user based security audit to verify service provider security services. Several research efforts have been proposed to engage users to verify cloud security properties based on the blockchain. The authors [Hao et al.(2018)Hao, Xin, Wang, Jiang, and Wang] proposed a remote audit approach using Proof of Work (PoW) as a distributed consensus mechanism. This approach can bring several benefits, it reduces collusion and cheating among Third party Auditors (TPAs) and minimizes user's overhead. However, every peer in the blockchain network participates in the audit process and this would minimize audit efficacy.

Most early studies such as [Lu et al.(2020)Lu, Zhang, Shi, Kumari, and Choo] utilizes Hyperledger Fabric to design a data integrity audit approach to verify files integrity in untrusted environment. The user can assign two private groups of nodes as TPAs to conduct the security audit. In addition, they designed two selection algorithms to select the right TPA that meets user requirements. Next, since most public audit schemes have a serious problem in certificate and key management, [Yang et al.(2020)Yang, Pei, Wang, Li, and Wang] developed a public audit scheme based

on blockchain, it provides traceability , support for dynamic hash table and resistance against a malicious auditor. Moreover, logs processing and analysis is the key method to monitor and detect incident. This research [Wang et al.(2019a)Wang, Peng, Tian, Chen, and Lu] proposed a public auditing scheme where a Third Party Auditor (TPA) is able to check the accuracy of the service provider logs using Merkle hash tree, only the root node is stored on the blockchain. This will reduce the computation cost on the service providers and generate log tags that cannot be altered.

3.3 Runtime Security Monitoring and Enforcement

Runtime security enforcement enables cloud tenants to verify applications during run time to meet regulations and compliance standards. It also enhances user's trust in the cloud service provider. For this reason a considerable amount of literature has been published to discuss enforcing security property in the cloud. The authors [Madi et al.(2016)Madi, Majumdar, Wang, Jarraya, Pourzandi, and Wang], [Majumdar et al.(2015)Majumdar, Madi, Wang, Jarraya, Pourzandi, Wang, and Debbabi] proposed a method to check cloud security compliance of access control policies and then discover security violations using formal methods. However, their approach can detect violation after it occurs and this can impact the data confidentiality and integrity. Another thread of research studies monitoring network traffic. Previous research [Dastjerdi et al.(2012)Dastjerdi, Tabatabaei, and Buyya], [Emeakaroha et al.(2010)Emeakaroha, Brandic, Maurer, and Dustdar], [Trapero et al.(2017)Trapero, Modic, Stopar, Taha, and Suri] modeled monitoring capabilities, proposed a framework, developed a security SLA approach to quantify security SSLA measurements, and identified circumstances that might cause SLA violations.

Multi-tenant cloud faces several security challenges that need to be addressed. Thus, the authors [Rios et al.(2017)Rios, Rak, Iturbe, Mallouli et al.] presented an enhanced approach for MUSA security assurance [MUSA(2015)] where monitoring agents are installed in the virtual machine to calculate the security metrics based on

network and system agents. The agents passively monitor the traffic and combine several events to capture incidents enforcing SSLA. The other system agents monitor active system process , memory and buffer size to detect abnormal behaviour.

3.4 SSLA verification in cloud

Over the last decade we have witnessed a significant revolution in cloud services including security services provided by gigantic tech companies. These security services vary from threat detection, Firewall management and cryptographic services. The service provider signs SSLA with customers to receive the security services as agreed in the SSLA agreement. This agreement explains the key characteristics of the cloud security service and initiates a common understanding for services providers and customers.

In order to verify SLA cloud services, many SLA enforcement approaches have been proposed.The authors [Silva et al.(2019)Silva, Silva, Rocha, and Queiroz] developed a model to calculate and measure service trust relying on security measurements. In their work, each security measurement is assigned a risk value to measure the service provider's trustworthiness in SSLA. The authors [Liu et al.(2020)Liu, Xia, Wang, Zhong, and Li] proposed a cloud behaviour SSLA model based on temporal logic to express SSLA as a series of properties and constraints instead of a fixed measurable values, then they applied model checking using a verification tool called UPPAAL to verify the cloud service compliance with security SSLA. To better validate SLA , a probabilistic model checker precisely using Continuous Time Markov Chain (CTMC) is presented [Krotsiani et al.(2017)Krotsiani, Kloukinas, and Spanoudakis], first a WS agreement language model is extended to include SLO. Second, using PRISIM they are able to detect the probability of having SLA violations on data confidentiality in regards to specific asset. One of the security requirements in cloud SSLA is resource isolation and layer consistency, this work [Madi et al.(2018)Madi, Jarraya, Alimohammadifar, Majumdar, Wang, Pourzandi, Wang, and Debbabi] examines this

property by applying Constraint Satisfaction Problem (CSP) to verify this property. First, the Isolation rules are represented in First Order Logic(FOL) to express each entity variable and relationships relying on cloud real data. The ultimate goal is to pass these rules through a solver to examine whether the constraint is satisfied or not to detect any violations.

Proactive verification of service level agreement is another new thread of research to address SLA dynamic changes and the detection of SSLA violations that might occur in the future. The authors in [Nawaz et al.(2018)Nawaz, Janjua, Hussain, Hussain, Chang, and Saberi] proposed an event driven methodology to specify a certain reaction to violation events once a violation is predicted to happen. In order to adapt probabilistic reasoning and predict future events, they divided their approach into two levels, the first level is to discover SLA discrepancies and then predict what will occur by applying state constraints which are rules to define each SLA metric quantified during specific time. They also rely on effect constraint rules to correlate each metric transition to another state to predict SLA violation.

Another work designed a proactive approach called LeaPS [Majumdar et al.(2019)Majumdar, Tabiban, Jarraya, Oqaily, Alimohammadifar, Pourzandi, Wang, and Debbabi] to automatically study system logs of events dependencies at run time. The adaption of Bayesian probability network is very useful to examine cloud system logs in live and test cloud environments.

CHAPTER 4: Proof of Encryption: Enforcement of Security Service Level Agreement for Encryption Outsourcing

4.1 Introduction

With the fast development and deployment of cloud and edge computing, a large portion of data processing and storage operations are actually outsourced to various types of service providers. For example, according to Forbes, about 77% of enterprises have at least one application or a portion of their enterprise computing infrastructure in the cloud. Service providers and cloud customers often use Service Level Agreement (SLA) to determine the committed resources or responsibilities [Sfondrini et al.(2015)Sfondrini, Motta, and You]. Since service providers often charge end users based on the amount of resources that they use (e.g.CPU cycles, network bandwidth, and memory), verification mechanisms have been designed so that claims from service providers can be verified [Akter and Whaiduzzaman(2017)] [Giachino et al.(2016)Giachino, de Gouw, Laneve, and Nobakht].

In parallel to the proliferation of cloud computing is Security-as-a-Service (SaaS) [Khettab et al.(2018)Khettab, Bagaa, Dutra, Taleb, and Toumi], [Sun et al.(2015)Sun, Nanda, and Jaeger]. Here end users depend on third parties to scan their network traffic, hard drive, or even execution states in systems to detect vulnerabilities and defend against on-going attacks. However, SaaS faces a serious challenge, which is the verification of services that are actually delivered. While the resources in a traditional SLA such as CPU cycles can be roughly audited based on the computation speed [Zhou et al.(2015)Zhou, Zhang, Yu, and Guo], the execution of security SLAs cannot be verified based on only the detection results. Below we describe an example. Assuming that an end user out-sources data storage to a service provider.

The Security SLA requires that data must be encrypted by a symmetric encryption algorithm not weaker than AES-256. Restricted by available computation resources and remaining energy, the data source cannot conduct encryption by itself. Therefore ,it needs to outsource the operations. However, the service providers have motivations to use a weaker encryption algorithm (or even store just plain text) to reduce the computation cost and electricity bill. Please note that this challenge is different from the Proof-of-Retrievablity [Tan et al.(2018)Tan, Hijazi, Lim, and Gani] problem since we care about the format in which data is stored instead of whether or not the provider has it. We need a new approach to solve this problem.

In this chapter, we propose to design mechanisms to achieve proof of encryption (PoE). We assume that the end customer has relatively weak computation resources and cannot accomplish all encryption operations by itself (it may or may not have resources to encrypt selected data blocks). The service providers will encrypt data with a predetermined algorithm and key strength. To prevent a service provider from cheating, the end user needs to verify some of the encryption results. There are several expected properties, such as randomness during data selection and configurable overhead, of the PoE mechanisms. We propose two mechanisms based on whether or not the service requester (aka end user) knows the encryption key. Through both analysis and experiments, we evaluate the proposed approaches on detection probability, overhead, and robustness to false results. Our evaluation shows that the proposed approaches allow end users to achieve proof-of-encryption with high probability and configurable overhead.

The contributions of the chapter can be summarized as follows. Firstly, we identify that compared to other types of SLAs, it is fairly difficult to verify the execution of security SLAs. As a special example, we discuss the problem of proof of encryption. Secondly, we define the expected properties of PoE mechanisms. Thirdly, we design two mechanisms for PoE that allow end users to verify the encryption operations by

service providers. Last but not least, we evaluate the proposed approaches through both analysis and experiments.

4.2 Related Works

Since many users of cloud computing are middle size or small size business, they cannot afford a special security team for the companies. Therefore, outsourcing the security services becomes a natural choice. Several efforts have been made to pave the way for auditing of the services. For example, in [Luna et al.(2015)Luna, Suri, Iorga, and Karmel],the authors propose to firstly map security SLAs to Service Level Objects (SLO). Cloud service providers (instead of end customers) need to assess the fulfillment of the SLOs. Similarly, in Cloud Trust Protocol (CTP) [CloudTrust Protocol Working Group(2015)], the cloud providers build an open API set to enable end customers to query providers about the security level of their services. Another factor that the authors in [Luna et al.(2015)Luna, Suri, Iorga, and Karmel] identify for security SLA enforcement is standardization. From this perspective, ISO/IEC 19086 [EC Cloud Select Industry Group (C-SIG)(2014)] and EU's General Data Protection Regulation (GDPR) [Erkuden Rios and Eider Iturbe and Xabier Larrucea and Massimiliano Rak and etc.(2019)] could become pioneers. Cloud companies often pay close attention to data storage encryption. For example, Concentra Health agreed to a settlement of $1.7 million for failing to meet industry-standard data encryption expectations [HIPPA(2015)].Another thread of research in this domain is the establishment of ontology for security feature understanding [Lee et al.(2015)Lee, Kavi, Paul, and Gomathisankaran]. This scheme provides a formal framework for representing knowledge for potential logic inference. While the approach allows end users to understand the security agreements with a provider, its auditing efforts focus on compliance with federal regulations, instead of fulfillment of security operations. Based on the analysis, we can see that there is not much effort in enforcement of security SLAs in cloud computing environments, especially for the fulfillment by cloud

providers. More research is needed to solve this problem.

4.3 The Proposed Approach

In this section, we will present the details of the proposed approaches. We will first analyze the expected properties of a Proof-of-Encryption mechanism. We will then describe the working procedures of the protocols. We will also analyze the detection capability of the proposed approaches.

4.3.1 Expected Properties of Proof-of-Encryption Algorithms

Randomness: The goal of the algorithm is to prevent a service provider from violating the SSLA by selectively encrypting only a portion of data, using a weaker encryption algorithm, or skipping encryption at all. To avoid incurring too much overhead at the end user, it can only verify encryption results of a small portion of data. Therefore, the selection procedure must be robust against pre-computation or guessing attacks. In other words, a service provider cannot predict the segments of data that the requester will challenge.

Robust against encryption on the fly attack: Since the requester will challenge the service provider and the response must be calculated based on both the challenge and the data encryption results, the approach must be robust against encryption on the fly attacks. In other words, the requester must be able to identify whether or not the encryption operation is conducted only after the provider receives the challenge.

Configurable overhead: Since a service requester usually has very limited computation and communication resources, the proposed approach needs to support configurable overhead. Quantifiable analysis needs to be provided to maintain balance between detection accuracy/capability and its commitment to resources for verification.

Support changes to data: This property is directly related to the data applications. For example, if a small portion of data is updated, the PoE operations should not be heavily impacted. Otherwise, the adoption of the approach will be greatly restricted.

From the description above, we can see that the first two properties are used to prevent the service provider from cheating. The last two properties focus on the obstacles of adoption. These properties are not bound to any specific encryption algorithms or cloud architectures. Therefore, the proposed approach can be applied to various scenarios.

4.3.2 System Assumptions

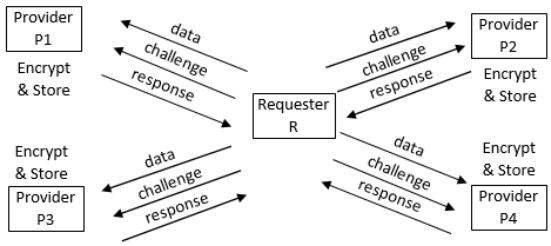


Figure 4.1: Application scenarios of PoE approaches.

Figure 4.1 shows the application scenario of the proposed approaches. We assume that a service requester R will outsource the storage of data to one or multiple providers P_i, $i = 1 \cdots s$. Here the providers P_i may belong to the same owner or multiple owners. Restricted by available computation resources, R cannot afford to encrypting all data with a strong algorithm by itself. It has to outsource the encryption operations to P_i. Please note that R has concerns on data confidentiality for attackers who can compromise the storage devices of P_i. Therefore, it needs to verify that P_i actually accomplishes the encryption of all data with the agreed algorithm and key strength as defined in SSLA. To achieve that, R will generate random challenges and P_i has to respond to them based on the challenges and encryption results.

Since the verification of data encryption happens between the service providers and data owners, we need to differentiate between two cases: whether or not the service

requester R has enough information (about the key) and hardware/software resources to encrypt selected data blocks by itself. In the first case, if the requester R knows the encryption key used by Pi and has the CPU power to encrypt some data with the key (it may not be power efficient for R to encrypt all data), the PoE problem becomes straightforward. The two parties R and Pi need to verify that: (a) they know the same information (the encryption results), and (b) the information is not generated on the fly. A variation of the real-time data flow verification scheme [15] can be adopted to achieve the goal.

However, in real life under many cases, the storage providers will not share the encryption keys with the requester. Therefore, even if R has a powerful CPU and corresponding resources, it cannot verify the cipher-text directly. We must design a different approach to the problem. In this chapter, we propose a cross-comparison based scheme to detect a dishonest service provider. The attacker is a dishonest service provider that tries to save encryption operations through violating the SSLA. It may have a superfast computer that can encrypt data on the fly. However, to allow such attacks, the attacker must be able to transmit data from storage to the computer through a link with very short delay. Similarly, if multiple service providers are involved, they may try to collude to cheat the requester.
The following table summarizes the symbols.

4.3.3 Proposed Mechanism for Scenario 1

In scenario 1, we assume that both the service provider P_i and the requester R know the encryption key. At the same time, R has the software/hardware resources to accomplish encryption on a selected group of data blocks. Therefore, the requester can randomly select a group of data blocks and challenge P_i for encryption results. In this way, the property of "randomness" is satisfied automatically. At the same time, since R can determine the amount of data that it will challenge P_i with, it can control the committed resources.

Table 4.1: Symbols used in the chapter.

R	service requester (end user)
P_i	service providers
$h(x)$	secure hash function known to both parties
m_j	jth data message sent by R to P
L	each data chunk contains L data blocks
t	length of specific patterns in encryption results to trigger verification operation
q_j	the probability that Provider P_j skips encryption of a data block
$p_{j,k}$	the percentage of data blocks provided to P_j that are also provided to P_k

The major challenge is to prevent P_i from encrypting the data blocks after it receives the request (encryption on the fly). To differentiate the situation in which encryption is pre-accomplished from the case in which data is encrypted after the request is received, we can measure the network delay between the request is submitted and the response is received. However, a dishonest provider may choose to hide the computation time in the network transmission delay. For example, based on BearSSL [16], on an Intel Xeon CPU 3.1GHz, AES-256 CBC mode can encrypt about 125MB/sec. Based on this measurement result, for a 1KB data block, the service provider needs only 8μsec to accomplish the encryption. It can easily hide the delay under the round trip communication time oftens of millisecond between R and P_i.

To detect this cheating behavior, we propose the following approach. Assume that the data blocks that P_i needs to encrypt are represented as m_j, j= 1 to n. These data blocks are divided into chunks with the size L. For example, m_1 to m_L form Chunk 1, m_{L+1} to m_{2L} form Chunk 2, so long and so forth. The corresponding cipher-text blocks are represented as c_1, c_2,$\cdots$,c$_n$. Within each chunk, CBC (cipher blockchaining) mode is used for data encryption. The initial vector(IV) for encryption can be jointly determined by R and P_i. In this way, we guarantee that within a chunk,

multiple blocks cannot be encrypted in parallel.

When the requester R wants to verify that proper encryption operation has been applied to the data blocks, it will randomly choose one block $m_{aL+j}(a = 0 \cdots \frac{n}{L})$ in a chunk and provide a random number r as the challenge. The service provider P_i needs to return the MAC (message authentication) code $hash(r,hash(r,c_(a_{+1L}),c_(a_{L+}j)$ $c_{aL} + 1))$. We can see that the hash result covers the first block, the last block, and requested block in the chunk .

If P_i has properly encrypted all data blocks and stored them, it just needs to concatenate the corresponding cipher text and return the results. On the contrary, should P_i have violated the SSLA and stored the data in other formats, it has to re-encrypt the blocks with the secret key determined in the SSLA. Since CBC mode is adopted, it cannot use parallel computing to accelerate the procedure. It has to encrypt the blocks one by one. Since the hash result puts $c_{(a+1)L}$ as the first entry, the hash calculation cannot start until the encryption of the whole chunk is accomplished. This operation will drastically increase the response delay.

4.3.3.1 Analysis

The chunk size L provides a user configurable parameter that can impact the increases in response time should the service provider conduct encryption-on-the-fly attacks. The larger is L, the longer time Pi needs to encrypt the whole chunk. In this way, R can adjust the choice of L based on the network connection quality and speed between it and Pi. Please note that the goal of initial vector and chain mode is to prevent an attacker from conducting random encryption of the data. It is not bound to AES or any other specific encryption function as long as the overhead can be enforced.

Another manipulation Pi can play is to store only the ciphertext of the first and last blocks of each chunk. This operation is not attractive to the service provider because of the following reasons. Firstly, to calculate and store the cipher-text of the

last block in a chunk, it needs to accomplish the encryption of the whole chunk since CBC mode is used. From this point of view, even if Pi decides to store the data in other formats, the cost of encryption has already been paid. Secondly, since the challenge of verification covers the cipher text of c_{aL+j}, if Pi stores only c_{aL+1} , it needs to re-encrypt the data blocks to get the value of c_{aL+j}. The introduced delay could still be detected by the requester.

4.3.4 Proposed Mechanism for Scenario 2

In real life scenarios, very frequently the service provider of encryption and storage and the service requester will belong to different organizations. Therefore, they will not trust each other on encryption key generation and storage. For example, should the data confidentiality be compromised because of key disclosure, it will be very hard to determine accountability if multiple parties know the key. Therefore, more frequently we need to deal with the scenario in which the requester does not know the encryption key.

Since R does not know the encryption key, it cannot directly challenge P_i for data result. Therefore, we need a new approach to randomly select the data blocks/records and verify their encryption results. In this approach, we assume that the requester R chooses symmetric encryption such as AES 128 for data encryption. Without losing generality, we assume that R will choose three different service providers P_1, P_2, and P_3 to accomplish the encryption and storage tasks. Some of the data blocks that R sends to the three parties will overlap so that R can cross compare the encryption results. In addition to R and P_i, we also assume that a certificate authority (CA) exists in the system and all four parties trust it.

4.3.4.1 Overview of Approach

In this part, we will provide an overview of the proposed approach. More details will be provided in subsequent parts. First, R will notify P_1, P_2, and P_3 that they are

chosen for its encryption and storage task. Each of them will work with CA to get a new public/private key pair. Please note that this new key pair is associated with a pseudo ID of each party. In this way, through looking at only the public keys, they cannot figure out the real identities of the service providers.

After getting the public/private key pairs, the service providers will run a multi-party Diffie-Hellman key generation protocol [Bresson et al.(2001)Bresson, Chevassut, Pointcheval, and Quisquater] with the help from R. Again all communications among the providers will go through R. In this way, they cannot figure out the real identities of each other. At the same time, the authenticity of exchanged information is protected by digital signatures with the new private keys.

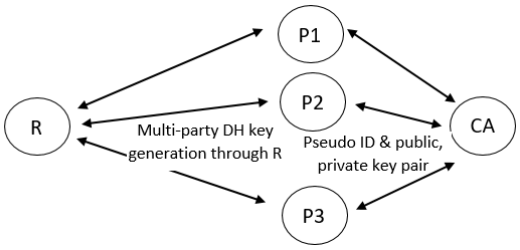


Figure 4.2: Overview of the approach for scenario 2.

Once the key generation procedure is accomplished, R can distribute different chunks of data to different providers. Since the providers know only there are other service providers but not their identities, they cannot collude to manipulate the encryption results. Since the requester R does not know the encryption key, it cannot directly verify the encryption results. However, it can cross compare the encryption results from multiple parties since they use the same key. R can either challenge different service providers with the same data blocks or use another method to randomly choose data blocks. An overview of the approach is shown in Figure 4.2.

4.3.4.2 Detailed Description

Step 1 : Generation of Encryption Key

As shown in Figure 6.2, R decides to choose P_1, P_2, and P_3 to help it encrypt data. It will notify them and also provide the identity of the CA. Now each provider will

contact CA and acquire a new public/private key pair for its temporary identity P_1', P_2', and P_3'. Please note that the temporary IDs are not linked to their real IDs. Once receiving the new public key certificates, P_1, P_2, and P_3 will share them with R. R will then distribute the certificates to all providers. At this time, only R and CA know the real IDs of the providers.

R will choose a finite cyclic group G of order w and a generating element g in G. R will distribute the generator g and the large prime number P to all three providers. They will then run the multi-party Diffie-Hellman protocol. Specifically, each provider will generate its own random number $r_{P_i'}$, calculate $g^{r_{P_i'}} \bmod P$, and send it to R. The integrity of the information will be protected by the digital signature associated with the new temporary ID.

As shown in Figure 6.2, all providers will send their digitally signed shares to R. R will then forward them to other parties that have not added their shares. Throughout the procedure, the requester R serves as the center of communication and transmits packets between the providers. In this way, a provider cannot learn the identities of other providers and collude with them. At the end of the key generation procedure, each provider will learn the group key $g^{r_{P_1'} \times r_{P_2'} \times r_{P_3'}} \bmod P$. The key is protected by three digital signatures from the providers. The requester R, although passes all traffic, cannot learn the group secret.

Step 2 : Selection of Challenged Data Blocks

In this part, we will introduce two methods to select data blocks for encryption result challenge. In the first method, R will send the same group of data blocks to two or even more encryption providers. These blocks will blend into other groups of blocks that are sent to them. Since the providers do not know the identities of each other, they cannot conduct off-line collusion. After confirming that the data blocks have been encrypted, R will challenge two or more parties with the same group of data blocks for encryption results. During the challenge-response procedure, R needs

to measure two parameters. First, it needs to measure the delay between the request is sent and the result is received. This is to prevent P_i from conducting encryption-on-the-fly attacks. Second, R needs to cross compare the encryption results of the same group of blocks from different providers. If a provider has chosen to violate the SSLA and uses other encryption algorithms or weaker keys, its encryption results will be different from those of others. Therefore, R will be able to verify the encryption results even if it does not know the key.

Letting R select data blocks for challenge, although achieving the randomness property, has several limitations. First, the distribution of the verified data blocks is very unbalanced. If a group of blocks are chosen, thousands of consecutive blocks will be verified. On the contrary, if a group of blocks are not selected, non of them will be verified. Second, the data access delay could vary a lot depending on the storage medium. For example, after encryption, a provider may move the data to external hard drive, or under an extreme condition, a tape drive. Therefore, when a challenge is received, the provider may need an extended period of time to get data back into memory. This may lead to increases in false alarms.

To solve these problems, we need to design a mechanism that selects data blocks for challenges when the encryption algorithm is running. In this way, the data has not been moved to external devices. At the same time, the selection procedures cannot be controlled by service providers. A potential solution is the criteria that have been used for mining in BlockChains. Assume that the requester R and provider P_i have determined the encryption algorithm, encryption mode, and key strength. R will now choose a pattern in the encryption result of a data block. If the result satisfies the pattern (e.g. the first t bits are all '0'), P_i needs to report the index of the block to R. This mechanism has the following advantages.

First, Randomness of Selected Data for Verification

Since the criteria are applied to the encryption results, the provider P_i cannot

predict which blocks to encrypt beforehand. The usage of initial vector (IV) prevents P_i from predicting the encryption results based on similarity of data blocks. From this point of view, the selection procedure is random. Even the requester R cannot predict which blocks will be selected. A provider may choose to lie about the encryption results or the index of the blocks. However, there are other providers and it does not know to which provider which blocks are shared. Therefore, should it choose to lie, it will face the risk that the same block may be encrypted by another provider.

Second, Configurable Overhead

Through adjusting the criteria pattern, R can control the probability that any block is challenged. For example, we can approximate any probability p with the sum of a group of negative powers of 2:

$$p = a_0 \times \frac{1}{2^0} + a_1 \times \frac{1}{2^1} + a_2 \times \frac{1}{2^2} \cdots + a_l \times \frac{1}{2^l} \cdots ; \tag{4.1}$$

Here the probability p has the value between [0, 1], and the value of a_i is either 0 or 1, depending on the probability we want to approximate. The accuracy of approximation is determined by the length of the encryption results.

4.3.4.3 Analysis of Detection Accuracy and Overhead

In this part, we will analyze the detection capability and overhead of the proposed approaches. We will discuss two scenarios when the provider P_i is benign or malicious, respectively. If P_i is benign, it will encrypt all data blocks based on the SSLA. Therefore, the only factors that impact the frequency of data verifications are the choices of encryption algorithm, the initial vector, the key, and the data blocks.

Without losing generality, we assume that when the first t bits of the encryption results are all '0', it will trigger the verification procedure. For each block, the probability that it does not trigger verification is $(1 - \frac{1}{2^t})$. Here we propose to divide

data into windows with the size of L consecutive blocks. If at least one block in a window satisfies the criteria, we say that the whole window is verified. Therefore, the probability that no block in v_t consecutive windows triggers verification is $(1-\frac{1}{2^t})^{v_t \times L}$.

Here we will follow the adjustment practice of blockchain mining. To prevent P_i from cheating, we require that if v_t windows of consecutive blocks do not trigger any verification operations, we will reduce t by 1 bit. In other words, we double the probability that a block satisfies the verification criteria. Similarly, if u_t consecutive windows have all triggered the verification procedure, we will increase t by 1 so that fewer blocks will be verified by R. Therefore, the requester R can adjust the choices of L, v_t, and u_t to control the probability that a block is verified.

If the provider P is malicious, it needs to find a trade-off between the percentage of data that it encrypts and the probability that it is detected by the requester. Assume that the provider has the probability q to not encrypt a data block. Since it does not encrypt the block, it will not report the index to the requester even if the encryption result could have triggered verification. Since for each block, the probability that it triggers verification is $\frac{1}{2^t}$, the probability that the provider misses the block is $q \times \frac{1}{2^t}$. Note that if another provider encrypts this block and discovers that the encryption results satisfy the criteria, it will report to the requester and this violation of SSLA will be detected. Therefore, the probability that in x consecutive blocks the malicious provider does not miss any blocks that could trigger verification is $(1 - \frac{q}{2^t})^x$.

In real life the situation will be more complicated. It is possible that two providers both skip the block. Therefore. even if it could have triggered verification, none will report to the requester. Let us assume that for Provider P_1 the parameters have the values (q_1, t_1) (P_1 has probability q_1 to not encrypt a block, and the trigger pattern covers the first t_1 bits.) Similarly, for provider P_2 the parameters have the values (q_2, t_2). Without losing generality, we assume that $t_1 \geq t_2$. In other words, if the encryption results satisfy the pattern of provider P_1, it will also satisfy the pattern of

P_2.

For any data block that R provides to both P_1 and P_2, the scenarios in which a violation of SSLA by either party is detected can be described as follows: (1) P_1 chooses to encrypt the block but P_2 decides to skip it; or (2) P_2 chooses to encrypt it while P_1 skips it. Now let us examine their probability.

If P_1 encrypts the block and the results satisfy its pattern but P_2 skips the block, the probability is

$$(1 - q_1) \times q_2 \times \frac{1}{2^{t_1}}; \tag{4.2}$$

Here since $t_1 \geq t_2$, the encryption results will automatically satisfy the pattern of P_2. Under this case, P_2's violation of SSLA will be detected.

On the other hand, if P_2 encrypts the block and the results satisfy its pattern but P_1 skips the block, the probability is

$$q_1 \times (1 - q_2) \times \frac{1}{2^{t_2}}; \tag{4.3}$$

Here we face a problem. An encryption result that satisfies the pattern of P_2 may not satisfy the pattern of P_1. To differentiate between the two cases, we need the provider P_2 to send back not only the index of the block but also the encryption result. In this way, we can tell whether or not the violation of P_1 can be detected. So the probability that P_1's violation can be detected is:

$$q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}}; \tag{4.4}$$

When we investigate the equations, we can learn the following information. First,

since the requester R does not know the encryption key, it can detect an SSLA violation only when the same data block is sent to more than one provider and their encryption results differ. Therefore, the higher percentage of blocks that are sent to multiple providers, the better chance to detect violations. If R can afford it, it should provide the same data blocks to as many providers as possible. This also assists the enforcement of data randomness and the prevention of collusion between providers.

Second, although R could choose different patterns for different providers (e.g. t_1 '0's for provider P_1 and t_2 '1's for P_2), the analysis above shows that R has the best chance of detecting violations when the patterns of different providers are similar. At the same time, the detection probability is determined by the longer pattern between two providers. On one side, the longer is the pattern, the lower chance that the encryption result triggers verification. Therefore, the detection probability will be low. On the other side, the longer is the pattern, the lower communication volume between the providers and requester. Therefore, it makes sense to keep the pattern length of different providers close to each other. In this way, we reduce the communication overhead without impacting the detection capability.

Third, we want to analyze the relationship between the detection capability and the value of q. In other words, what is the best strategy of a dishonest service provider to avoid detection. If we look at only Equation (2), for P_2 to avoid being detected, it needs to reduce the value of q_2, and increases the value of q_1. In other words, it needs to encrypt more data blocks and expects P_1 to skip more. Similarly, if P_1 wants to protect itself from being detected, it needs P_2 to increase q_2. If we jointly consider the sum of the two detection probabilities, we get $(q_1 + q_2 - 2 \times q_1 \times q_2) \times \frac{1}{2^{t_1}}$ Analysis shows that the joint detection probability has the double-saddle shape. The probability is low when both q_1 and q_2 are very close to 0 or 1. In other words, if we look at only P_1 and P_2, they can avoid detection by either encrypting all data blocks or encrypting none at all. For the latter option, however, they will be caught

very soon since the trigger pattern will become very short (thus many blocks should trigger verification). If there is at least one honest service provider in the system, it will report data blocks satisfying the criteria and the dishonest ones will be caught as well.

4.3.4.4 Managing the Probability of Sharing Blocks

The analysis in Section 6.3.5 considers only the data blocks that have been submitted to both P_1 and P_2. In real applications, restricted by the cost, only a part of data blocks will be submitted to more than one encryption service provider, as shown in Figure 4.3. Without losing generality, we use the probability $p_{1,2}$ to represent the ratio between the number of blocks that are known to both P_1 and P_2 and those known to only P_1. Therefore, for the requester to detect an SSLA violation by P_1 through the report of P_2, the probability is

$$p_{1,2} \times q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}}; \tag{4.5}$$

If P_1 skips the encryption of v data blocks, the probability that it is detected by R is

$$1 - (1 - p_{1,2} \times q_1 \times (1 - q_2) \times \frac{1}{2^{t_1}})^v; \tag{4.6}$$

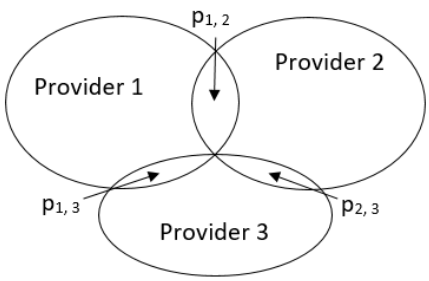


Figure 4.3: Only data blocks submitted to multiple providers can be used for detection of SSLA violations.

4.4 Implementation and Experimental Results

In this part, we will conduct experiments to collect quantitative results about the proposed approaches. The experiments include both data encryption and access operations on real networked devices and the simulation of the impacts of different parameters on the detection capability.

4.4.1 Detection of Encryption-On-the-Fly Attack

As we describe in Section 2.3.3, if a dishonest service provider chooses to encrypt data blocks after receiving a verification request, it will introduce extensive delay into the response procedure. To verify that we can detect such attacks through measuring the network delay, we choose three could service providers and implement encryption functions on a virtual machine hosted by them. A client machine will issue a verification request. If the virtual machine has already accomplished data block encryption, it will reply with the hash results as described in Section 2.3.3. Otherwise, if the virtual machine has not encrypted the data blocks, it will have to conduct encryption and hash in sequence. We choose two sites, one on-campus and one off-campus, as the client to conduct the experiments. The delays are measured in four consecutive days during both day time busy hours and late in the night. The data blocks that are challenged have the size of 1K Byte. The window size is 10,000 blocks. Five groups of experiences are conducted and their average delay is shown in Figure 4.4.

From the figures, we can see that the on campus connections with the service providers are very stable. If the virtual machines have to encrypt the 10M Byte data on the fly, it will take somewhere between 80 to 100 ms. The increases in delay obviously deviate from the normal network conditions. A user will be able to detect such changes. We assume that a dishonest service provider will choose to use the customized environments such as AES-NI [Gueron(2012)] to avoid detection. Based

on [Pornin(2018)], the AES 256 CBC mode encryption speed will increase about 4 times if AES-NI is adopted, which will bring the extra delay down to 20 to 25 ms. Such increases are still detectable compared to the normal measurement results.

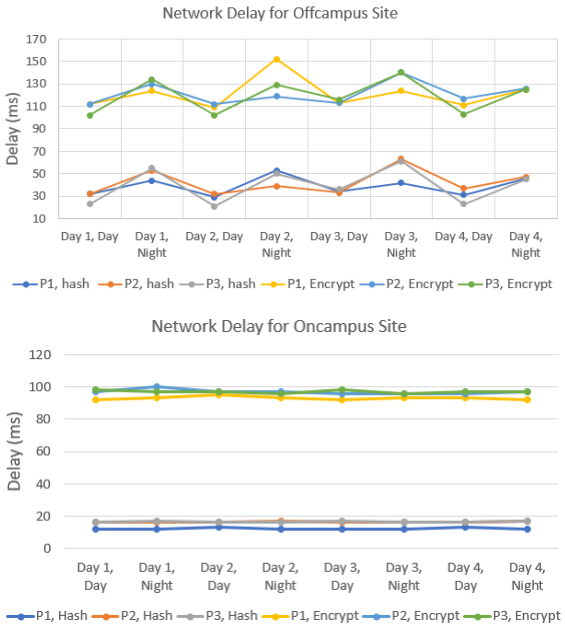


Figure 4.4: Detection of on-the-fly encryption attacks through changes in network delay.

4.4.2 Cross-Comparison Detection of Symmetric Encryption Results

data sharing probability $p_{i,j}$ between two providers	ratio b/w data at each provider and overall data volume
0%	33.3%
10%	37%
20%	41.6%
30%	47.6%
40%	55.5%
50%	66.7%

In the second group of experiments, we will use simulation to investigate the impacts of different parameters on the detection capability of the requester when it does not have a copy of the encryption key. To simplify analysis, we assume that the requester chooses three encryption service providers. The probability $p_{i,j}$ represents the

percentage of data blocks of provider P_i that are also known to provider P_j. Please note that as $p_{i,j}$ increases, the probability that a dishonest provider is detected also increases. As the price to pay for improved security, each provider has to encrypt more data blocks. The table above shows as $p_{i,j}$ increases from 10% to 50%, the changes of encryption load at each provider. We can see that when the sharing probability reaches about 30%, each provider needs to encrypt about half of overall data blocks.

The simulation results are shown in Figure 4.5. Here we experiment with the data block sharing probability between two providers $p_{i,j} = 10\%$ and 20%. The probability that a provider skips encrypting a data block ranges from 5% to 15%. In other words, a service provider will randomly pick 1 to 3 data blocks to skip encryption in every 20 blocks. The length of the pattern in the encryption results to trigger data verification ranges from 7 bits to 13 bits. In other word, on average, one block in every 128 to 8192 data blocks could trigger a verification operation. In all six sub-figures, on the X-axis we show the number of data blocks that are provided to a provider and it needs to encrypt all of them. On the Y-axis, we show the probability that a dishonest provider is caught if it skips encryption of some data blocks. Please note that the number of blocks on X-axis is roughly in logarithmic scale.

From the figures, we can see that as the number of data blocks increases, the probability that a dishonest service provider is caught increases very fast. When R provides only 10% of the data blocks of the provider P_1 to $P2$ and P_1 skips only 5% of the encryption operations, when the number of data blocks reaches 500,000, P_1 has 20% chance to be caught. Please note that if we choose the data block size to be 1K Byte, that is only 500M Byte data that R sends to P_1. Considering the amount of data that is uploaded to the cloud everyday, this is a very small number. The service providers do not have a strong motivation to cheat when they are aware of the high probability of detection.

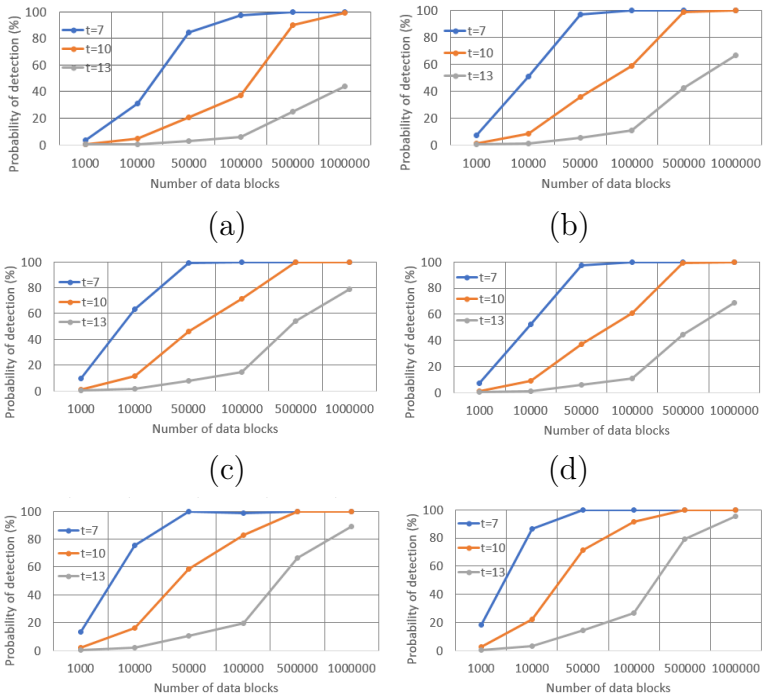


Figure 4.5: Detection probability as the number of data blocks changes. (a) $p_{i,j}$=10%, q_i=5%; (b) $p_{i,j}$=10%, q_i=10%; (c) $p_{i,j}$=10%, q_i=15%; (d) $p_{i,j}$=20%, q_i=5%; (e) $p_{i,j}$=20%, q_i=10%; (f) $p_{i,j}$=20%, q_i=15%.

Another feature we can see from the simulation results is that the length of the pattern in encryption results to trigger verification has a large impact on the detection capability. When we use a longer pattern, the probability that a data block will trigger verification decreases exponentially. An end user needs to maintain balance between the communication overhead and the detection capability when it outsources the encryption operations.

4.5 Discussion on Security of Approaches

In this part, we will discuss the safety of the approaches. We are especially interested in the following aspects.

Robustness against off-line collusion

Although the proposed approach in Section 4.3.4 tries to hide the real identities of the service providers during key generation, some of them may still have off-line

agreement to cover up for each other. For example, P_2 and P_3 may exchange the hash results of an encryption key to determine whether or not they are assigned to the same task. If the answer is 'yes', they can then identify the data blocks shared between them. However, the knowledge that the colluding parties can learn is limited. For example, if there is at least one provider that does not collude with them, they face the same probability of detection as we analyze in Section 6.3.5. To hide the information on data distribution among different encryption providers, R can involve more parties during the key generation procedures. After the key is determined, it will provide data blocks to only a subset of the providers. In this way, a dishonest provider cannot derive out the number of encryption servers solely based on the key generation procedures.

Verification of encryption algorithm vs. properties

The objective of POE approaches is to verify the execution of a specific encryption algorithm. Some may have the concern that the approach in Section 5.3 verifies the homomorphic property instead of the algorithm itself. Several methods can be used to solve this problem. The end user can analyze some features of the cipher text to verify the encryption algorithm. For example, RSA has the homomorphic feature. Based on the distribution of the cipher text, we can estimate the size of the product of the two large prime numbers. In this way, we can derive out whether or not the service provider uses smaller prime numbers in the algorithm. Similarly, there has been efforts [Xiao et al.(2019)Xiao, Hao, and Yao] to estimate the strength of encryption algorithm based on the features of cipher text.

4.6 Conclusion

In this chapter we investigate the problem of proof of encryption. Specifically, when a data source asks one or multiple providers to encrypt its data with specified algorithms and key strength, there must be a way for it to verify the execution of the SSLA. We investigate two scenarios: cross verification of duplicate data records

and homomorphic encryption based approach. The designed approaches can detect the violations of SSLA on encryption. We conduct both experiments and analysis to investigate the mechanisms and their probability to detect a dishonest service provider.

When we put the research problems of the chapter in a bigger view, the goal is to allow end users to verify the execution of security service level agreement (SSLA). Different from the SLAs that focus on resource usage aspects (e.g. CPU cycles, available bandwidth), security related SLAs are harder to verify since there is not always a security incident available for detection. We plan to design a comprehensive suite of mechanisms to cover more features in security enforcements. We are especially interested in the services such as malware scanning and effectiveness of firewalls. The completeness of the approaches can be assessed with an ontology framework.

CHAPTER 5: Proof of Outsourced Encryption: Cross Verification of Security Service Level Agreement

5.1 Introduction

The inevitable growth of cloud data requires service providers to create a transparent security services so users are able to guarantee their data safety. The verizon data breach report [Verizon(2020)] indicates that cloud services and applications represented 24% of overall last year data breaches. Although, the cloud services provide scalability and massive storage, recent research direction indicates that the service providers falls short not only in security but also in meeting ultra-low latency demands [Hui et al.(2019)Hui, Zhou, An, and Lin]. Given these circumstances a user based security verification mechanism remains a glimmer of hope for customers to verify the confidentiality of their data.

In this chapter, our objective is to propose Asymmetric mechanisms to achieve proof of cross-verification of outsourced encryption. In addition, Partially Homomorphic Encryption (PHE) provides a protection against data disclosure and enables user to send data stream among networks in ciphertext. In this approach, we adopt PHE to allow cloud users and providers to perform only one operation on the ciphertext either addition or multiplication instead of decryption the ciphertext so that the cloud user can verify encryption services. The proposed approach is different from the previous chapter symmetric problem, the cloud user selects only one service provider to negotiate homomorphic encryption instead of sending out verification data to multiple providers for verification. Through both analysis and experiments, we evaluate the correctness of the proposed approach using BAN logic and probability of detection to detect SSLA violations.

The contributions of the chapter can be summarized as follows. First, in the SSLA enforcement domain, we study the problem of proof of outsourced encryption using homomorphic encryption case. We verify the correctness of the approach using BAN logic. Then, we evaluate the proposed approaches through both analysis and experiments.

The remainder of the chapter is organized as follows. In Section 5.2, we describe related work that we can benefit from. In Section 5.3, we present the details of the security verification approach using Partially Homomorphic Encryption. We then verify the correctness of the approach using BAN Logic. Section 5.4 compares our approach to public auditing, and finally section 5.5 discuss the experiment results using parameters that can impact the detection of a dishonest service provider and analyze the detection probability. Finally, Section 5.6 concludes the chapter.

5.2 Related Works

Depending on the usage of outsourced data, end users may choose different encryption algorithms. For example, if the sole purpose of encryption is data confidentiality, symmetric encryption algorithm with decent key size may be the most power efficient choice. However, with the increasing need of operations on cipher text, homomorphic encryption has attracted a lot of research efforts [Fun and Samsudin(2016), Yang et al.(2019)Yang, Huang, Liu, Cheng, Weng, Luo, and Chang, Zhao and Geng(2019)]. The algorithms allow end users or service providers to directly operate on cipher text to get the results they want. In this chapter, we analyze both types of encryption algorithms and the methods to verify the encryption.

5.3 Proposed Mechanism for Homomorphic Encryption Based Verification

The mechanisms described in previous sections have a few drawbacks. First, submitting the same data blocks to multiple service providers will increase cost at the requester. Since the probability of detection is directly related to the percentage

of shared blocks, an end user has to submit more duplicate data blocks to multiple service providers to increase the detection capability. Second, when Blockchain based mechanism is adopted, even the requester cannot predict which data blocks will trigger verification. This uncertainty causes confusion at the requester. When no verification is triggered for an extended period of time, it cannot tell whether or not it is caused by an SSLA violation. The third drawback comes from the potential of service provider collusion. In a previous Section , the anonymity among service providers is achieved through a new pair of public/private keys with fake identity. In real life, however, considering the limited number of large scale storage service providers, it is possible that two providers maintain a stealth communication channel and collude to fool the requester. The two providers can compare the digest of the data blocks to identify the shared ones. They will then encrypt only those shared blocks.

To overcome these drawbacks, we investigate the outsourced storage service based on homomorphic encryption. A homomorphic encryption (HE) scheme allows computations to be performed on cipher text without the need for the cipher text to be decrypted. One of the original purposes of HE algorithms is to protect data privacy. While there are several ways to classify the HE algorithms [Chaudhary et al.(2019)Chaudhary, Gupta, Singh, and Majumder, Chen et al.(2019)Chen, Wu, Lu, and Ren, Fun and Samsudin(2016)], in this chapter we focus on the "Partial Homomorphic Encryption" (PHE) methods since they have weaker assumptions on the algorithm properties. Therefore, the designed approach can be generalized to "Somewhat Homomorphic Encryption" and "Fully Homomorphic Encryption" [Fun and Samsudin(2016)].

In PHE algorithms, we assume that either additive or multiplicative operations upon the plaintext group are supported (but not both). In [Fun and Samsudin(2016)], more than 10 types of PHE algorithms are discussed. Most PHE algorithms are

built upon asymmetric encryption schemes and have the non-deterministic property. Several of the PHE algorithms have been adopted in commercial systems such as CryptDB [Popa and Redfield(2011)], MyCrypt [Tetali et al.(2013)Tetali, Lesani, R.Majumar, and Millstein], Crypsis [Stephen et al.(2014)Stephen, S.Savvides, Seidel, and Eugster], and CMD [Gadepally et al.(2015)Gadepally, Hancock, B.Kaiser, Kepner, P.Michaleas, Varia, and Yerukhimovich].

Our designed approach can be applied to both additive and multiplicative homomorphic algorithms. Here we define the homomorphism as:

$$Enc(M_1 \odot_p M_2) = C_1 \odot_c C_2; \tag{5.1}$$

M_1 and M_2 are plain text, C_1 and C_2 are corresponding cipher text, and $\odot_p$ and $\odot_c$ are group operations in plain text and cipher text space, respectively.

Following the assumptions in Section 4.2, we assume that the service requester R has data records d_1, d_2, $\cdots$, d_l to encrypt. Here it needs to choose only one service provider P_1 and negotiates the homomorphic encryption algorithm with it. The service requester does not need to know the encryption key. When the data records are provided to P_1, the requester will add some noise records n_1, n_2, $\cdots$, n_w into the data for future verification operations. The generation of the noises will be described below.

P_1 will encrypt all data records with the selected algorithm. To reduce communication overhead, it will not transmit the encryption results back to the requester. However, for every L records, it needs to construct a Merkle's hash tree with the encryption results, as shown in Figure 5.1. The roots of the trees will be sent back to the requester as the digests of the encryption results.

Now let us explain how to embed noises into data to detect a dishonest service provider. Consider a simple example. Assume that $n_i \odot_p d_k = d_j$. Here d_j and d_k

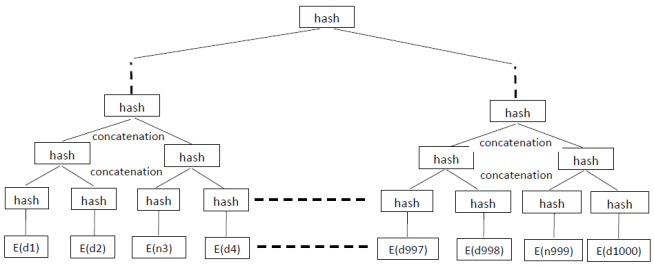


Figure 5.1: Merkle's hash tree for integrity verification of encryption results.

are two randomly selected data records. The noise record n_i will be inserted into data records and transmitted to P_1. If P_1 is honest, it will encrypt all the records (including the noises), generate the hash trees, and send back the tree roots.

When the end user R tries to verify the encryption operations, it will request the encryption results $Enc(n_i)$, $Enc(d_j)$, and $Enc(d_k)$ from P_1. Since the provider has committed the roots of trees during the encryption procedure, the end user can easily verify the integrity of the encryption results. It can then compare $Enc(d_j)$ to $Enc(n_i) \odot_c Enc(d_k)$. If the two results are equal, the end user confirms that the service provider conducts the encryption operations. Otherwise, a violation is detected.

For a dishonest service provider, since it knows that the end user can verify only the data records that satisfy the $\odot_p$ operations, it can examine the submitted data records and try to locate them. Generally speaking, for a noise record $n_i \odot_p d_k = d_j$, the provider P_1 needs to examine all possible data pairs d_k and d_j to identify them, which lead to the complexity of l^2. Should it successfully locate the data pair d_j and d_k, it can generate fake encryption results so that the $\odot_c$ function over cipher text is satisfied.

To defend against such attacks, the service requester can adopt the following methods. First, it can put the three data records n_i, d_j, and d_k into three Merkle's trees. In this way, the dishonest service provider has to search for the target data records within a much larger domain. By the time it identifies the last data record of interest, the encryption results of the other two records have already been committed. The second technique that the requester can use is to use more data records to construct

a noise record. For example, if we have $n_i \odot_p d_{k1} \odot_p d_{k2} \odot_p d_{k3} \cdots d_{kr} = d_j$, the complexity of identifying all involved data records will increase to l^r. The end user can choose the number of involved data records to overwhelm the computation capability of the dishonest service provider.

Below we provide some quantitative analysis on the relationship between the behaviors of the dishonest service provider and its probability of being detected. We assume that each Merkle's tree covers L data records. The dishonest service provider has probability p to apply the negotiated encryption algorithm to a data record. For the remaining records, it will use a weaker encryption algorithm. If the end user chooses r data records from the L records and applies the $\odot_p$ group operation upon them to construct a verification challenge, the probability that P_1's violation is detected can be represented as:

$$Detection(L, p, r) = \begin{cases} 1 - \frac{C_{L-r}^{pL-r}}{C_L^{pL}} & : r <= pL \\ 1 & : otherwise \end{cases}$$

If $L >> r$, the probability of detection can be estimated as 1- p^r. From the formula, we can see that if a dishonest provider tries to avoid detection, it needs to encrypt more data records. From the end user's point of view, it needs to use more data records to construct the challenge. Please note that here we assume the end user chooses all records from the same tree. It can choose the data records from multiple trees.

Compared to the cross verification mechanism described in Section 4.3, this approach has multiple advantages. First, it does not require the same data records to be provided to multiple encryption parties simultaneously. The end user can choose only one service provider. In this way, it can reduce the storage cost and avoid the collusion of multiple service providers. Second, the end user has full control over the

density and placement of the inserted noise data records. Since the end user can randomly select data records to construct the challenge but a dishonest service provider has to conduct exhaustive search, the computation workload at the attacker will be very heavy. At the same time, we do not have to worry about the unpredictability caused by Blockchain.

5.3.1 Correctness Verification of the Proposed Approach

In this part, we will formally analyze the correctness of the proposed approach based on the BAN logic. While BAN logic has been widely used for verification of authentication protocols [Gao et al.(2018)Gao, Deng, Wang, and Kong], it has also been used to verify the execution of service level agreement, especially the delivery of intended information to participating parties [Bhasker and Murali(2020),Pourpouneh and Ramezanian(2016)]. Our recent work in this domain focuses on protocol verification in smart health environments [Wang et al.(2019b)Wang, Shi, and Qin, Wang et al.(2020)Wang, Qin, and Wang].

Since our protocol is not an authentication protocol, the BAN logic cannot directly verify its safety. On the contrary, we focus on two aspects: (1) the service requester R can verify that the received message is what the provider P believes in, and (2) it has confidence in the received encryption results for subsequent SLA violation detection.

During our verification procedure, we use the 'message meaning rule', the 'nonce-verification rule', the 'freshness rule', the 'principal sees rule', and the 'jurisdiction rule', the meaning of which can be found in [Anderson(2008)]. We use some extensions to BAN logic that were presented in [Gope and Sikdar(2019)] to handle the combination of two components in a message. Our protocol safety partially depends on the large search space from which the provider P cannot identify the selected messages for homomorphism property. To simplify the analysis procedure, we design the following interaction operations. Here P represents the service provider and R is the service requester. K is the key that P selects for data encryption.

$R \longrightarrow P$: $(n, v1, m1, v2)$ (a)

$P \longrightarrow R$: $E_{P_{pri}}(n, h(E_K(v1)), h(E_K(m1)), h(E_K(v2)))$ (b)

$R \longrightarrow P$: challenge for $v1, m1$, and $v2$ with nonce n (c)

$P \longrightarrow R$: $E_{P\_pri}(n, E_K(v1), E_K(m1), E_K(v2))$ (d)

In this simplified version of the protocol, the real data block that R wants to encrypt is $m1$. $v1$ and $v2$ are embedded for verification purposes and we have: $v1 \odot_p m1 = v2$ and $Enc(v1) \odot_c Enc(m1) = Enc(v2)$. In real life, these values will be hidden in a large number of plain text blocks. The freshness of the protocol is protected by the random nonce n. We use the hash results of the cipher text to replace the Merkle's tree in Section 5.3. The messages are signed by the private key of P for authentication. To accomplish formal verification, we need to first convert the message to idealized representation. Here we use $\{\{m1'\}_K\}_{\{v1\}_K}$ to represent $Enc(v1) \odot_c Enc(m1)$ (which should also be $Enc(v2)$) in the protocol. Two reasons lead to this change: (1) we use it to represent the combination of $Enc(v1)$ and $Enc(m1)$ so that BAN rules can be applied to it, and (2) should P violate the protocol and not encrypt $v2$, the homomorphism no longer holds. Therefore, we use $m1'$ to represent the potential violation before we can verify it. The idealized protocol is illustrated below. In addition to the message, the two parties also share the knowledge about the plain texts and hash of the cipher texts.

Idealized protocol:

Msg: $P \rightarrow R$: : $\{ \{n, \{v1\}_K, \{m1\}_K, \{\{m1'\}_K\}_{\{v1\}_K} \}_{\{P\_pri\}}$

$P \overset{v1}{\longleftrightarrow} R$, $P \overset{m1}{\longleftrightarrow} R$, $P \overset{v1+m1}{\longleftrightarrow} R$,

$P \overset{h(E_K(v1))}{\longleftrightarrow} R$, $P \overset{h(E_K(m1))}{\longleftrightarrow} R$, $P \overset{h(E_K(v1+m1'))}{\longleftrightarrow} R$ },

Below we show the assumptions and verification goals of the protocol. Their meanings are explained in the brackets. The four goals guarantee that R will receive the intended messages from P and can conduct subsequent verification.

Assumptions:

A1:	R believes $\overset{P_{pub}}{\longrightarrow} P$	[R believes P has the public key P_{pub}]
A2:	R believes $R \overset{h()}{\longleftrightarrow} P$	[R believes it shares secure hash function *h()* with P]
A3:	R believes $\#(n)$	[R believes that n is fresh]
A4:	R believes $P \Rightarrow \{\}_K$	[R believes P has jurisdiction over $\{\}_K$]
A5:	P believes $\{v1\}_K$	[P encrypts *v1* with the key *K*]

Verification Goals:

G1: R believes $P \mid\sim \{m1\}_K$ [*R* believes that *P* once said $E_K(m1)$]

G2: R believes P believes $\{m1\}_K$ [*R* believes that *P* believes $E_K(m1)$]

G3: R believes $\{m1\}_K$ [*R* believes $E_K(m1)$]

G4: R sees both $\{m1\}_K$ and $\{m1'\}_K$ [*R* can cross verify the two values]

The verification procedure is shown in the following figure. All four goals are met. Here goal 4 allows the requester R to see two encryption results $\{m1\}_K$ and $\{m1'\}_K$ so that verification of the homomorphism property can be conducted through comparison of their values.

Verification procedure:

Msg: $P \rightarrow R$: : $\{\{n, \{v1\}_K, \{m1\}_K, \{\{m1'\}_K\}_{\{v1\}_K}\}_{\{P\_pri\}}$

$P \overset{v1}{\longleftrightarrow} R$, $P \overset{m1}{\longleftrightarrow} R$, $P \overset{v1+m1}{\longleftrightarrow} R$,

$P \overset{h(E_K(v1))}{\longleftrightarrow} R$, $P \overset{h(E_K(m1))}{\longleftrightarrow} R$, $P \overset{h(E_K(v1+m1'))}{\longleftrightarrow} R\}$,

// apply the message meaning rule to the message and A1, we have

(1) R believes $P \mid\sim (n, \{v1\}_K, \{m1\}_K, \{\{m1'\}_K\}_{\{v1\}_K})$,

// *R* believes *P* sends the message

// so we have

(2) R believes $P \mid\sim (\{m1\}_K)$ // *R* believes *P* sends $E_K(m1)$, Goal 1

// apply the freshness rule to A3

(3) R believes $\#(n, \{v1\}_K, \{m1\}_K, \{\{m1'\}_K\}_{\{v1\}_K})$, // R believes it is fresh

// apply the nonce-verification rule to (1) and (3)

(4) R believes P believes $(n, \{v1\}_K, \{m1\}_K, \{\{m1'\}_K\}_{\{v1\}_K})$,

// so we have

(5) R believes P believes $(\{m1\}_K)$ // *R* believes *P* believes $E_K(m1)$, Goal 2

// apply the jurisdiction rule to A4 and (5)

(6) R believes $(\{m1\}_K)$ // Goal 3

// following a similar procedure, we can get

(7) R believes $(\{v1\}_K)$

// apply the "principal sees" rule to (1)

(8) R sees both $\{m1\}_K$ and $\{\{m1'\}_K\}_{\{v1\}_K}$

// apply the "principal sees" rule to (7), (8) and A5

(9) R sees $(\{m1'\}_K)$ // Goal 4

□

5.4 Comparison to Public Auditing of Outsourced Storage

In Section 4.1, we shortly discuss the difference between the POE problem we study in this chapter and the POR (proof of retrievability) problem. In this section, we provide more discussion and investigate how we can benefit from research results in that area. We will focus on the results in public auditing for cloud storage since this direction represents the focus in past few years.

5.4.1 Achievements in Public Auditing of Cloud Storage

According to [Wang et al.(2010)Wang, Wang, Ren, and Lou], public auditability allows an external party, in addition to the data owner, to verify the integrity of outsourced data on cloud. Here the data is usually stored in the format that the owner provides. For example, the owner needs to encrypt data by itself if it wants the data to be stored in a more secure format. The storage provider will not apply other functions to the data. While the early approaches to POR [Juels and Kaliski(2007), Wang et al.(2009)Wang, Wang, Li, Ren, and Lou] support integrity verification by only data owners, a series of subsequent research achievements [Wang et al.(2010)Wang, Wang, Ren, and Lou, Tian et al.(2019)Tian, Nan, Chang, Huang, Lu, and Du, Tu et al.(2017)Tu, Rao, Huan, Wen, and Xiao, Wang et al.(2013)Wang, Chow, Wang, Ren, and Lou, Li et al.(2018)Li, Yu, Yang, Min, and Wu] have been designed to allow non-interest third party to accomplish the task. Several approaches to public auditing are built upon the homomorphic linear authenticator technique [Ateniese et al.(2007)Ateniese, Burns, Curtmola, Herring, Kissner, Peterson, and Song, Shacham and Waters(2008)]. They usually build a bilinear map upon two or three multiplicative cyclic groups. Using the hardness of discrete-log assumption, the data owner will construct signatures of the data blocks that contain information from both the data contents and their index numbers. The data blocks and signatures will then be provided to cloud storage provider. During auditing, the

third party will send out a challenge. The storage provider will use the challenge to calculate a linear combination of the data blocks. It will also use the signatures to calculate another part of the response. To prevent the verifier from combining multiple responses to compromise data privacy, the storage provider will also use a random number to achieve masking. When the verifier receives the responses, it will project them to the bilinear map and compare their equality to accomplish verification.

After the basic approach, multiple extensions have been designed. For example, in [Wang et al.(2013)Wang, Chow, Wang, Ren, and Lou] the authors adjust their original design of the masking function to provide better protection to data privacy. In [Tu et al.(2017)Tu, Rao, Huan, Wen, and Xiao], the authors remove data block index from the signatures and use Merkle Hash Tree to maintain the order information. In this way, the insertion, update, and removal operations to a block will not impact other blocks. Tian et. al. [Tian et al.(2019)Tian, Nan, Chang, Huang, Lu, and Du] extend the approach to fog computing environments. They design different signature generation algorithms for mobile sinks and fog nodes so that they can be transformed. Last but not least, Li et. al. [Li et al.(2018)Li, Yu, Yang, Min, and Wu] design a mechanism that supports users to update their signature keys for data blocks.

While the approaches described above have differences in multiple aspects, their strong points can be summarized as follows: (1) allow non-interest third party in addition to data owner to verify data integrity while preserving data privacy; (2) support unlimited number of times of verifications; (3) maintain low communication overhead since the signatures instead of data blocks are transmitted during verifications, and (4) support batch verification.

5.4.2 Challenges and Potential Improvement to our Approach

While the advantages summarized above for public auditing are very attractive, the differences between POR and POE (proof of encryption) problems make direct adoption of the techniques very difficult. Below we will describe the challenges caused by

these differences, and present potential improvements inspired by these mechanisms.

The most important difference is that in POR problem both data owner and cloud storage provider process data in the same format. For example, the signatures are generated based on the data that the cloud provider stores. In the POE problem, however, this fact no longer holds. Since the data owner cannot conduct the encryption function, it has to outsource the operations to external service provider. Should the service provider refuse to share the encryption key with the data owner because of security concerns, the data owner can only generate signatures of data blocks based on the plaintext instead of cipher text. At the same time, to prevent a malicious party from generating fake signatures and impersonating the owner, the signatures of data blocks must contain some secret information. This is represented as the secret key in the homomorphic linear authenticator in [Wang et al.(2010)Wang, Wang, Ren, and Lou].

In the POE problem, if we want to support non-interest third party to accomplish verification, we need a procedure similar to that of the public auditing. First, the verifier sends out a random challenge. The encryption service provider needs to calculate two parts of response based on the cipher text and signature of the plaintext, respectively. Note that the cipher text and signatures are generated with different secrets. To make sure that the two parts of response are equal, the encryption function and signature generation function need to be commutative (since the two secrets are applied in reverse order). Therefore, we cannot design a generic signature generation algorithm independent of the encryption mechanism (since they need to be commutative). This also presents a direction for future improvement to our approach.

From the discussion above, we can see that since in the proof of encryption problem the data owner and encryption service provider use plaintext and cipher text respectively, the design of a public auditing algorithm is very challenging. For some encryption algorithm, we may choose a commutative signature generation algorithm

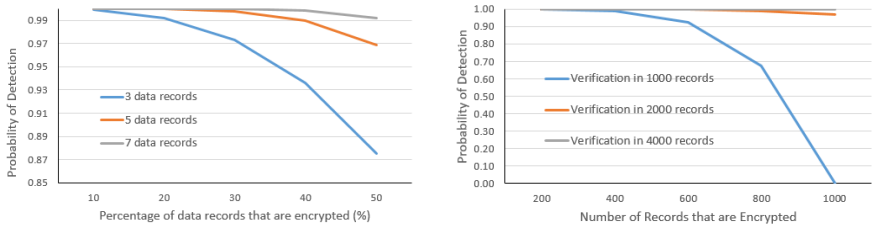
to accomplish the task. This will be our next target in research.

5.5 Implementation and experimental results

In this part, we will conduct experiments to collect quantitative results about the proposed approaches. The experiments include both data encryption and access operations on real networked devices and the simulation of the impacts of different parameters on the detection capability.

5.5.1 Homomorphic Encryption Based Detection

In this group of simulation, we investigate the relationship between the encryption workload at the service provider and its probability of being detected if it violates the SSLA. Here we assume that the end user requests the provider to construct a Merkle's tree for every 1000 encrypted data records. The service provider will randomly select data records for encryption with the probability p. For the remaining records, it will choose some weaker encryption algorithm. Therefore, if not all data records covered by a verification challenge are correctly encrypted, the end user will detect the violation.



(a) (b)

Figure 5.2: Detection probability of the homomorphic encryption approach.

In Figure 5.2.(a), on the X direction, we have the probability that a data record is encrypted with the negotiated homomorphic algorithm. We consider the cases that a verification challenge is constructed with 3, 5, and 7 data records, respectively. On the Y direction, we have the probability that the violation is detected by the end user when it challenges the results in one Merkle's tree. We can see that even when

the service provider encrypts half of the data records, the end user can construct a challenge and detect the violation with the probability close to 90%. Under this condition, the risk is too high for the service provider to conduct the attack.

In Figure 5.2.(b), on the X direction, we have the number of data records that are encrypted by the provider with the negotiated algorithm. We experiment with 1, 2, and 4 Merkle's trees, respectively. As the number of involved Merkle's trees increases, the percentage of encrypted data records decreases. We assume that the end user will choose 5 records to construct a verification challenge. On the Y direction, we have the probability that a violation is detected. For example, when the end user constructs the challenge with five records selected from 2000 records (2 hash trees) and the provider encrypts only 1000 of them, the probability of being detected is close to 97%. For the case that the end user selects five records from 1000 records and the provider encrypts 800 of them, the chance of being detected will still be about 70%. The detection probability will drop to 0 when the service provider correctly encrypts all data records (thus, there is no violation).

5.6 Conclusion

In this chapter we investigate the problem of proof of encryption. Specifically, when a data source asks one or multiple providers to encrypt its data with specified algorithms and key strength, there must be a way for it to verify the execution of the SSLA. We investigate two scenarios: cross verification of duplicate data records and homomorphic encryption based approach. The designed approaches can detect the violations of SSLA on encryption. We conduct both experiments and analysis to investigate the mechanisms and their probability to detect a dishonest service provider.

When we put the research problems of the chapter in a bigger view, the goal is to allow end users to verify the execution of security service level agreement (SSLA). Different from the SLAs that focus on resource usage aspects (e.g. CPU cycles,

available bandwidth), security related SLAs are harder to verify since there is not always a security incident available for detection. We plan to design a comprehensive suite of mechanisms to cover more features in security enforcements. We are especially interested in the services such as malware scanning and effectiveness of firewalls. The completeness of the approaches can be assessed with an ontology framework.

CHAPTER 6: Proof of Network Security Services: Enforcement of Security SLA through Outsourced Network Testing

6.1 Introduction

With fast development and wide adoption of cloud computing, lightweight startup becomes a practical approach. Since such small or mid-size companies often have very few technicians, they often need to outsource the security functions such as Deep Packet Inspection (DPI) [Sahay et al.(2019)Sahay, Meng, and Jensen]. Similar to other cloud based services, the end customers and security service providers need to sign a Security Service Level Agreement (SSLA) to concrete the details of the services and price. For example, the service provider commits to continuously update its network security rule sets so that within 48 hours of the discovery and publicizing of any malicious payload, it will be able to detect and block it.

While such security SLAs sound very attractive to end users, there is one challenge that must be overcome: verification of the execution of the SLA. Specifically, end users need to verify that the SSLAs are actually in effect. While this sounds reasonable, it is fairly hard to enforce when we consider the specialty of the SSLAs. Let us consider an example. Customer u orders the DPI service from service provider s. In the SSLA s commits to scan at least 99.99% of network traffic to u to protect it from malicious attacks. As the business of u becomes better, more customer traffic is attracted to the site and s can no longer keep its promise. A wrong yet viable solution for s is to reduce the amount of traffic that it scans for u. Please note that the misbehavior of s has a good chance of escaping detection since it is possible that the skipped packets do not contain any malicious contents. This scenario is quite different from those performance SLAs [Hermanto et al.(2019)Hermanto, Iskandar, Hendrawan, and

Edward, Samuels et al.(2017)Samuels, Syambas, Hendrawan, Edward, Iskandar, and Shalannanda] in which a customer can detect degradation in performance. Lack of verification for SSLA deserves more research efforts since many end users now depend on third parties to secure their networks.

In this chapter, we will focus on the verification of outsourced network security services. We do not restrict our approach to any specific security services. On the contrary, we just assume that s needs to examine all traffic to u based on a set of rules. In our approach, when the SSLA between u and s is in effect, u will bring in a non-interest third party t to help it test the coverage and effectiveness of the security services. Please note that this is different from the scenario in which u hires a separate company to conduct a thorough pen-testing [Casola et al.(2020a)Casola, Benedictis, Rak, and Villano] since such operations usually have relatively high cost and need to involve multiple parties including the service provider s. To accomplish this procedure, we assume that u knows the details of the security rules. Therefore, it will be able to use our verification infrastructure to test the services that s provides. For example, it can craft a group of packets so that our infrastructure can send them to u from different nodes. Based on the packets that reach to it, u can verify the enforcement of the SSLA.

While the basic idea is straightforward, several problems must be properly addressed before our infrastructure can be deployed. First, since our infrastructure will send network traffic, mechanisms must be designed to prevent it from being abused for network attacks. At the same time, corresponding credential and log information must be maintained for the testing requests. Second, mechanisms must be designed to prevent our infrastructure from faking the network tests. Only in this way can u differentiate the packets being discarded by s from those not transmitted from our infrastructure. Third, our infrastructure must recruit a large number of distributed nodes to prevent s from identifying the testing packets. Last but not least, the veri-

fication procedure must provide configurable balance between the detection accuracy and overhead.

The contributions of the chapter can be summarized as follows. First, our analysis shows that one thorough pen-testing is not enough to enforce the SSLA for network security. Unpredictable, user initiated verification will serve the purpose better. Second, we propose an infrastructure through which an end user can test its network security services with configurable frequency and self-crafted test cases. The integrity and authenticity of the test are properly protected. Third, we analyze the detection capability and overhead of the proposed approach.

6.2 Related Work

In this part, we will describe the state-of-the-art research in several directions from which we can benefit. We are especially interested in the establishment and enforcement of the security SLA for customers. At the high level, researchers have tried to build frameworks for the definition, description, and enforcement of security SLAs. For example, in [Trapero et al.(2017)Trapero, Modic, Stopar, Taha, and Suri], the EU researchers built a framework SPECS [Rak et al.(2013)Rak, Suri, Luna, Petcu, Casola, and Villano] that allowed users to prepare, negotiate, implement, and remediate security SLAs. The responsibility of SLA monitoring is put on the cloud provider. In [de Carvalho et al.(2017)de Carvalho, de Andrade, de Castro, Coutinho, and Agoulmine], the authors summarize several systems based on different standards [Luna et al.(2015)Luna, Suri, Iorga, and Karmel]. The efforts in [Casola et al.(2020b)Casola, De Benedictis, Rak, and Villano] try to embed security into the system from the design phase. They emphasize at both component and application levels. But they do not discuss the enforcement in the system execution procedures. The MUSA project of EU [Rios et al.(2016)Rios, Mallouli, Rak, Casola, and Ortiz] adopts a similar design concept and tries to enable the deployment of cross-platform applications.

To enable the definition and negotiation of the security SLAs, a common language that can be understood by both service providers and customers is essential for the system. In [Casola et al.(2017)Casola, De Benedictis, EraÅcu, Modic, and Rak], the authors extend the WA agreement [Andrieux et al.(2007)Andrieux, Czajkowski, Dan, Keahey, Ludwig, Nakata, Pruyne, Rofrano, Tuecke, and Xu] so that in addition to the description capability, the approach can also select corresponding components for the automatic execution of the security SLAs. In [Kaaniche et al.(2017)Kaaniche, Mohamed, Laurent, and Ludwig], the authors design the secrSLA language to define the security agreements. In addition to identifying the security rules in SLAs, the customers also need to determine the measuring parameters. In [Wonjiga et al.(2019b)Wonjiga, Rilling, and Morin], the authors identify the key performance indicators (KPI) and extend the CSLA language [Kouki and Ledoux(2012)] to define the monitoring SLAs. To improve user experiences in SLA definition, an interactive system was presented in [Casola et al.(2016)Casola, De Benedictis, Rak, and Rios] for SLA generation.

After the definition of security SLAs, the next step is the measurement of the parameters so that SLA violations can be detected. Different from several other approaches that depend on the cloud service provider to measure performance indicators, the authors of [Hussain and Al-Mourad(2014)] specify that third party could be used for SLA violation detection. In [Casola et al.(2017)Casola, De Benedictis, EraÅcu, Modic, and Rak], the approach links security properties with mechanisms through constraints so that violations can be detected. Example approaches are built to measure parameters for network security [Wonjiga et al.(2019b)Wonjiga, Rilling, and Morin] and data integrity [Wonjiga et al.(2019c)Wonjiga, Rilling, and Morin].

The definition and enforcement of security SLAs also benefit from the advances in the new techniques such as AI and blockchain. For example, in [Wonjiga et al.(2019a)Wonjiga, Peisert, Rilling, and Morin], the authors propose to add hash

results of data into blockchains to generate secure and unforgeable data records while preserving user privacy. Similarly, the research group applied blockchain to data integrity protection [Wonjiga et al.(2019c)Wonjiga, Rilling, and Morin] so that both end users and cloud providers can verify the results. In [Rios et al.(2016)Rios, Mallouli, Rak, Casola, and Ortiz], data mining technique is used for deep packet inspection so that network IDS can be enforced.

In parallel to the advances in definition and enforcement of SLAs are changes in penetration testing for enterprise security. With the fast increases in cyber attacks upon cloud computing environments, corporations often refer to internal or external penetration testing to evaluate their network defense capabilities. This service is often outsourced to security vendors that provide independent evaluation reports. Previous work [Al Shebli and Beheshti(2018), Denis et al.(2016)Denis, Zena, and Hayajneh] discussed automated penetration testing frameworks, techniques and tools commonly used while conducting this rigorous test. In addition, latest research [Schwartz and Kurniawati(2019)] shows that penetration testing could benefit from the dramatic resurgence of artificial intelligence and the availability of massive cyber security data sets.

Penetration testing is not a one-time deal. According to Cybersecurity Maturity Model Certification (CMMC), a cooperation should run the test one to two times per year. Since a thorough test can often impact the normal operations of a company, continuous and user-initiated tests may serve the purpose better. This demand leads to the design and experiment of our approach.

6.3 Proof of Network Security Services: Proposed Approach

In this section, we will present the details of the proposed approach. We will first discuss why penetration testing through a third party vendor is inadequate for SSLA enforcement. We will then analyze the expected properties of a Proof-of-Network Security Services mechanism. We will describe the working procedures of the protocols

and how to prevent external verifier from cheating. We will also analyze the detection capability of the proposed approach.

6.3.1 Inadequacy of Penetration Testing

Penetration Testing is the procedure of conducting controllable network scanning and attacks in order to discover vulnerabilities in systems and networks. Since the tester usually conducts controllable scanning and attack, the impacts are not catastrophic. Based on the recommendation of CMMC, a cooperation needs to conduct one to two thorough pen-testing each year. However, because of the wide deployment of Software Defined Networks and data centers, the frequency of network configuration changes is much higher. For example, the data in [Foerster et al.(2019)Foerster, Schmid, and Vissicchio] shows that the inter-data center networks can change their connections every few minutes. Since the network changes will affect the firewall rules, a more flexible and targeted test mechanism is needed.

Another problem is that penetration testing cannot really verify the execution of the security SLAs. For example, when the network security and firewall rules are originally configured, they faithfully represent the requirements of the SSLAs. Therefore, the penetration testing results will support this conclusion. However, as the system runs, the conflict between security rules and performance demand may appear [Shah et al.(2019)Shah, Patel, and Jinwala]. While the performance SLAs (such as CPU cycles and number of network requests served) can be easily verified by end users, the security SLAs are hard to tell. An end user cannot tell the difference between an undetected attack from a scenario in which no attack exists. Therefore, cloud providers have an intention to disable security measures to improve system performance. The mechanism that we design can capture such intentional negligence.

6.3.2 Expected Properties of Proof-of-Network Security Services

In this chapter, we propose an end-user initiated, non-interest third party executed verification mechanism for network security SLA. Specifically, an end user can ask a third party to test outsourced network security services with a group of carefully crafted packets. It can verify both the execution of the tests and their results. While the goals are intuitive, a more formal description of the expected features will allow us to compare our approach with existing mechanisms. Below, we will describe the properties that our approach can deliver.

Randomness of Testing Activity: If the security service provider knows that the end user may conduct random test to verify the execution of SSLA, it will try to identify such traffic and handle it differently so that SSLA violations cannot be detected. To prevent this from happening, the testing activities must be random. In other words, the frequency, time, format, and content of the test cannot be predictable to the service provider.

Sender Unpredictability: In our approach, we assume that the security service provider s is dishonest about the provided services. Therefore, it has the motivation to detect and disguise any third party security testing activities. In this way, the source from which the testing packets are sent must be unpredictable to the service provider. Please note that some of the security testing is stateful and multiple rounds of packet exchanges are expected. Therefore, the source node cannot use fake IP address during the testing procedure.

Authenticity of Testing Request: In our approach, the third party will send out network testing packets to u upon the user's request. If not managed properly, this network testing service can be abused by malicious parties to conduct real attacks. Therefore, thorough verification must be carefully conducted to make sure that the requester and the testing target are the same node. The authenticity of the testing

traffic must be verifiable as well.

Verification of Execution of the Tests: In our approach, the tester t is not trustworthy either. Therefore, it is possible that the tester does not send out the security testing packets at all. From the end user's point of view, it cannot differentiate this scenario from the scenario in which all testing packets are successfully filtered out by the service provider. Therefore, corresponding mechanisms must be designed to prevent a tester from not serving the request.

Based on the discussion above, we can see that these properties serve different purposes. The first two properties are used to prevent the service provider from identifying the testing traffic and handling it differently. In the third property, we try to prevent the testing service from being abused for malicious attacks. The last property focuses on protection to the end user. These properties are not bound to any specific encryption algorithms or network architectures. Therefore, the proposed approach can be applied to various scenarios.

6.3.3 System Assumptions and Attacker Model

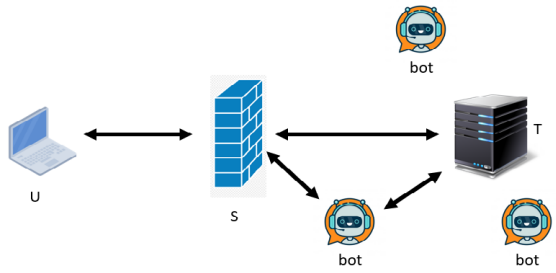


Figure 6.1: Application scenarios of the proposed approach.

Figure 6.1 shows the application scenario of the proposed approach. We assume that an end user u uses the network security services provided by s. u wants to use a third party to verify whether or not s is satisfying the security service level agreement (SSLA). Therefore, it resolves to the tester t. To simplify the scenario, we assume that u carefully crafts a group of packets and asks t to conduct the test. To prevent s from recognizing the source address of t and the testing traffic, t can recruit a group of *bots* to actually conduct the test. The test results can then be shared with u. The scenario involves four parties: u, s, t, and *bot*. Just as u cannot fully trust s, it cannot fully trust t either. Therefore, it needs to verify the execution of the testing procedure as well.

The service provider s follows a "not risk being caught" model. In other words, if the server can satisfy all SLAs without threatening its profit, it will follow all the rules. However, if it knows that it can violate the security SLAs without being caught, it will skip some of the security operations to improve system performance. Different servers may have different threshold values for risks of being caught. In our approach, the user u can adjust the frequency and granularity of tests to prevent a server from cheating.

The tester t and *bots* also follow the "not risk being caught" model. If they know that the user can verify whether or not the network testing is conducted, they will follow the request strictly to conduct the test and send back the results. However, if

the user cannot tell whether or not the test is conducted, they will cheat as well.

The following table summarizes the symbols we use.

Table 6.1: Symbols used in the chapter.

pub_x/pri_x	public/private key pair of node x
$cert(x)$	public key certificate of x
$h(x)$	secure hash function known to all parties
$E_k(m)$	message m encrypted with the key k
pkt_j	jth packet for security testing

6.3.4 Proposed Approach

In real life scenarios, after the user u and server s finalize the details of the security SLA, a thorough test may be conducted to verify that all rules in the SLA are functioning. After that, the user u may randomly select several rules in the SSLA and request a non-interest third party to conduct a second round test. This test can be conducted at any time, by any node, and in any format as long as the SSLA is still in effect.

6.3.4.1 Overview of Approach

We assume that u selects a group of rules in the SSLA to test. It will carefully craft a group of packets that can be used to test the effectiveness of the rule set. After constructing the packets, it will contact the tester t and send the packets to it. Note that application level encryption can be used between u and t so that the server s cannot learn the details of the test even if it monitors the network traffic of u. After receiving the request and verifying the authenticity of the packets, t will choose a *bot* to actually conduct the network testing. The *bot* will send out the packets constructed by u and record the results of the test. At the same time, u may receive some of the packets sent by *bot* based on the construction of the packets and the security rules in effect. Finally, the *bot* will report the results to t, who will then share the results with u. u can combine the packets that it receives from the *bot* and the records shared by t to determine whether or not the selected rules are in effect. An overview of the

approach is shown in Figure 6.2.

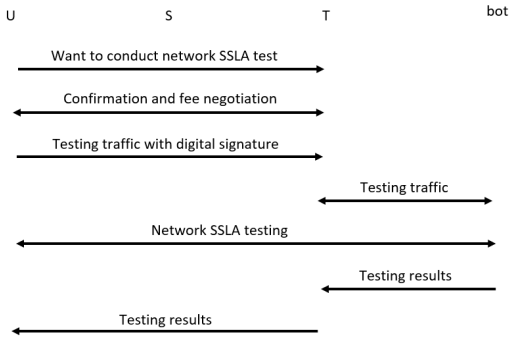


Figure 6.2: Overview of the approach.

6.3.4.2 Detailed Description

Step 1 : Negotiation of Testing Request

When the user u plans to conduct a network test, it will first contact node t. Since t provides the network testing services, its identity and public key certificate can be acquired. u will generate a testing request with its private key and t's public key.

$$u \rightarrow t : (u, t, cert(u)); \tag{6.1}$$

$$u \rightarrow t : (u, t, E_{pub_t}(E_{pri_u}(u, t, request, r_1, hash(u, t, r_1)))); \tag{6.2}$$

Here $cert(u)$ represents the public key certificate of node u, and r_1 represents a random number. Since the request is protected by two layers of encryption, the tester t will be able to verify the authenticity of the message. The two parties t and u can then generate a session key k through a secure method such as a variation of Diffie-Hellman. During the generation procedure, t will use the IP address in the network testing packets as the communication destination to verify the mapping between the IP address and the certificate. Note that even though the service provider s recognizes the tester t, it cannot figure out the details of the traffic because of data encryption.

Once u and t establish a secure session key, u can send the carefully crafted testing packets to t. u will construct a Merkle's hash tree based on the contents of the packets

and sign the root of the tree with its private key. The packet structure is shown below.

$$u \rightarrow t : (u, t, E_k(u, t, pkt_1, pkt_2, \cdots, pkt_n), \\ E_{pub_t}(E_{pri_u}(u, t, hash\ tree\ of\ the\ packets))); \tag{6.3}$$

Step 2 : Conduction of the Test

After receiving the testing request and verifying its authenticity, t will recruit a *bot* to conduct this test. Here we use the *bot* to refer to a machine that has collaboration with t and they will split the fee that u pays. t can run the operations as a P2P network or a crowd-sourcing service [Su and Pan(2016)]. Once identifying the *bot*, t will forward the request, packets, and digitally signed hash tree to it so that it can verify authenticity of the test as well. The *bot* will then send the packets to u, and interact with it based on its replies to the packets. *bot* will maintain a record of the interaction history and share the record with t and u later.

Step 3 : Results of the Test

After the test is accomplished, the *bot* will generate the record and the complete interaction history. It will then submit the history to t, who will then provide the record to u. During the whole procedure, the IP address of the *bot* is not visible to the service provider s. If u is happy with the submitted results, it will keep a record and use the results as a proof to s for any violations to SSLA.

6.3.4.3 Analysis of Expected Properties

In this part we will analyze how the approach satisfies the expected properties and defends against several attacks.

First, Randomness of Testing Activity

During the SSLA verification procedure, only u and s have the complete view of the network security rules stated in the SSLA. Therefore, it is easy for u to craft packets

for a test to the rules. Based on [Wool(2010)], a modern firewall can easily contain hundreds of rules and thousands of objects. Therefore, the number of combinations of rules and objects that u can choose for testing is large. The probability that s can correctly identify the incoming or outgoing testing traffic through random guess is really low.

Second, Sender Unpredictability

In our approach, we clearly differentiate the node t who manages the testing services and the node *bot* who actually executes the test. From the service provider s' point of view, it can easily identify the IP address of node t. However, since t can recruit a *bot* node to actually conduct the test, the packet sender cannot be identified by s.

Some readers may still have the concerns about identifying the testing traffic based on the receiver of the packets. For example, when s monitors traffic and finds out that u contacts t, it assumes that u will start third party testing. Although it cannot figure out the IP address of the sender *bot*, it can reactivate all network security rules of node u so that it can successfully pass the test. This action, however, will not guarantee the avoidance of detection should s actually violate the SSLA. Below is the analysis.

The test that u and t negotiate can be conducted at a future time. For example, u and t may determine that the test will be executed every seven days. Since the communication between t and u is encrypted, s could not figure out the determined time and format of testing. If s decides to reactivate the SSLA for u, the objective of u is achieved since it only wants its network security to be protected. Unless s keeps all SSLA of u active all the time, a violation could still be captured once s deactivates the rules.

Third, Authenticity of Testing Request

Since the network testing traffic may contain suspicious or even malicious packets, the tester t and sender *bot* must verify the authenticity of the request to prevent the

mechanism from being abused for network attacks. In our approach, t and *bot* can verify the digital signature of the request from u. At the same time, Merkle's hash tree of the packets is constructed and signed by u as well so that the contents of each packet is authenticated. When *bot* sends out a packet, an authentication method such as the keyed MAC code can be attached to the packet so that later t and *bot* can prove that they strictly adhere to the testing request.

Fourth, Verification of Testing Activities

In our approach, the tester t and user u do not have mutual trust. Therefore, after paying t for the test and submitting the packets, u must have a method to verify that the test actually happens. For example, if u crafts a group of packets that will all be discarded by s should all SSLA are enforced, u will not be able to differentiate the scenario of an honest service provider from that of a cheating *bot* who does not send out any test traffic. Therefore, some method must be designed to prevent such events from happening.

To identify a dishonest t or *bot*, u can intentionally embed a group of packets that can successfully penetrate the SSLA into the request. When they are mixed with the real testing packets, t or *bot* cannot differentiate them since they do not know the details of the security rules. Therefore, if the *bot* does not send out the testing packets, u will not receive any packets that could have penetrated the network security rules. Under this case, u can detect the dishonest *bot*.

6.3.5 Analysis of Detection Accuracy and Overhead

In this part, we will analyze the detection capability and overhead of the proposed approach. We will discuss two scenarios when the service provider s is benign or malicious, respectively. Without losing generality, we assume that the SSLA that s needs to enforce for u contains R rules. For each test, u will randomly select r rules and construct the testing traffic. To make sure that *bot* will actually conduct the network test, it will also embed e testing packets that will go through the SSLA and

reach to u. We assume that for the testing of each rule, the user u needs to provide b bytes to t. The total amount of traffic between u and *bot* for testing of one rule is B bytes. Here $B \geq b$ since some test cases involve multiple rounds of interaction.

Now let us look at the first scenario in which s is honest and executes all SSLAs faithfully. When u decides to conduct a third party test of the network security rules, it will first communicate with t and provide $(r + e)b$ Bytes of data to t. t will then share the packets with the selected *bot*. Please note that at this time we ignore the network traffic and computation overhead caused by the generation and transmission of the digital signatures and hash trees. Once the testing traffic and *bot* are selected, u will undergo the test. Depending on the actions that s should take on the testing packets, u and *bot* may exchange up to $(r + e)B$ bytes of traffic. During this procedure, s will check the network security rules for u for each packet. So the total communication overhead at u is $(r + e)(b + B)$ Bytes.

The second scenario is more complicated. Let us assume that out of the R rules u and s have for their SSLA, s is enforcing only f rules so that less resource will be used. Therefore, when u randomly selects r rules for testing, it is possible that some violations will be detected.

Since there are R rules between u and s altogether, the different combinations of r rules are selected for testing is $\binom{R}{r}$. Similarly, the number of combinations that all r rules under test are selected from the f rules enforced by s is $\binom{f}{r}$. Therefore, for a single round of detection, the probability that at least some violation is detected is: $1 - \frac{\binom{f}{r}}{\binom{R}{r}}$. As an exception, if $r > f$, the probability that s is caught is 1.

So the detection probability depends on three parameters: the total number of rules between u and s, the number of rules that s enforces, and the number of rules that t tests. The next figure illustrates the detection probability when the three parameters change their values. In Figure 6.3, the total number of rules between u and s is 300. s enforces 10% to 70% of the rules. u will select 3 to 15 rules for testing. Since u has

a large free space to select the rules to test, it becomes extremely hard for the server s to guess which set of rules to enforce. As Figure 6.3 shows, even when s enforces up to 70% of the rules and u selects only 3 rules to test, u still has about 65% chance to catch the violation in one round of test. Please note that the communication overhead for u stays roughly the same no matter s is dishonest or benign.

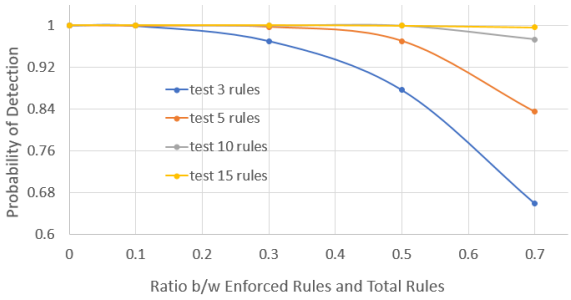


Figure 6.3: Detection capability of the approach.

6.3.6 Reducing False Alarms

The analysis results in Section 6.3.5 show that if u and t successfully accomplish the negotiation about testing and the packets from *bot* successfully reach to s, we have a very good chance to detect the violations. Here we need to consider a scenario in which false alarms may arise. Since *bot* will send out a group of packets to test the network security functionality, it is possible that some of the packets will look 'suspicious' to the routers or network security devices along the path and get discarded (just as many spam emails are silently removed by the routers).

These silent operations by the routers may lead to false alarms of the proposed approach. For example, if a router discards all of the testing packets, u will assume that the *bot* does not execute the contract and request refund from t. As another example, if a router discards the packets that should have been identified and removed by s because of the SSLA, u will wrongly label s as an honest service provider.

To reduce false alarms caused by such scenarios, we propose to use the method described in [Kanich et al.(2008)Kanich, Kreibich, Levchenko, Enright, Voelker, Paxson, and Savage]. Specifically, when t is helping users to test their network security

functions, it can recruit multiple *bots* to execute the test. The goals here are two folds: (1) The probability that the same subset of testing packets are removed is low when they travel along different paths, thus reducing false alarms. (2) Based on the feedback from u, t can learn the actions of routers along different paths and assign testing requests to different *bots* accordingly in the future.

6.4 Quantitative Results

In this part, we will present some quantitative results about the proposed approach. The experiments focus on the detection capability of the approach and the impacts of traffic filtering by the network devices.

6.4.1 Detection of Time-Varying SSLA Enforcement

As we describe in Section 6.3.5, even when the service provider disables the enforcement of only 30% of the network security rules, it still risks the chance of 65% of being detected. From this point of view, partially enforcement of the network security rules is not an attractive method to s.

In this part, we consider another option that s could adopt to avoid detection. We know that the service provider faces severe challenges between performance and security when the network traffic volume is high. Therefore, s could choose to bypass a certain portion of network traffic to solve the problem. This scenario is different from the method described in Section 6.3.5 since now a part of the network traffic will experience thorough examination, while the remaining part will go free.

Similar to the scenario studied in Figure 6.3, we assume that there are 300 rules between u and s, and u randomly chooses 2, 4, 6, 8 or 10 rules to test. To simplify the analysis, we assume that for each rule u will need only one packet to test. We assume the probability that s decides to bypass a packet without examination is p. This probability can be applied to each testing packet. The following figure shows the detection capability of the proposed approach.

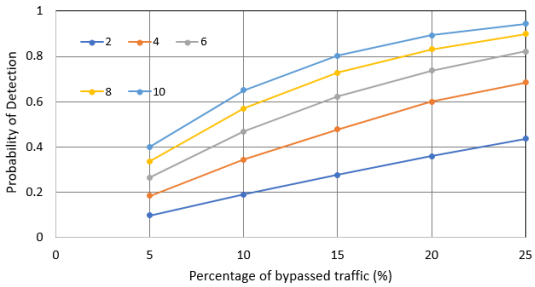


Figure 6.4: Detection capability under traffic bypass.

In the figure, we assume that s will bypass 5% to 25% of the network traffic. If a testing packet is not examined, it will reach to u and u will detect the violation. Out of all the selected rules, u needs to capture only one skipped packet to identify the violation. From the figure, we can see that with the increase of the number of tested rules, the detection probability reaches above 80% even when s skips only 25% of the network traffic. Note that the number of tested rules in this figure is independent of the total number of rules. This property is especially beneficial for large corporations with complicated network security regulations and many security rules.

6.4.2 Mitigating Impacts of Packet Removal along the Path

With the development of network security practise, devices at both edge and core networks start to enforce tight packet inspections to filter out potentially harmful traffic as soon as it enters the network. From this point of view, when a *bot* sends out the network test traffic upon the request of u, it is possible that some of the packets are labeled as 'malicious' and silently discarded by routers. If this scenario happens and u does not receive the packets, false alarms will arise.

To reduce such false alarms, one mechanism that u and t can adopt is to choose more than one *bot* from the network and expect that different network devices will adopt different security measures when they examine the traffic contents. This also supports the discussion in previous sections on the recruitment of *bots* from different areas of the network. When multiple groups of testing packets choose disjoint paths, different subsets of packets will reach to s. Should s disable some or all of the security

rules, we have a better chance to catch the violation.

Figure 7.5 shows the quantitative results. Here we refer to the idea in [Desai(2012)] and classify the testing packets into three categories based on their sensitivity to security devices: low, middle, and high. Without losing generality, we assume that for each type of packets, the probability that an edge router of an Autonomous System (AS) discards the packets silently is 0.1% (for low), 1% (for middle), and 5% (for high), respectively. We also assume that once a packet enters an AS, it will not be examined again. Therefore, if a testing packet travels through 5 ASes before it reaches to s, it will be examined for 5 times.

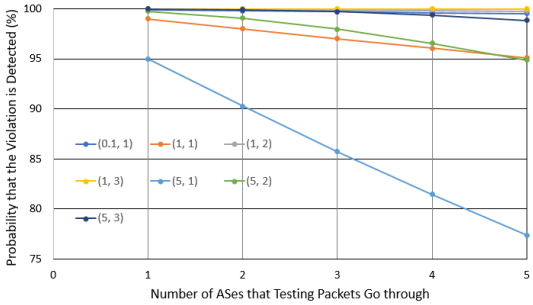


Figure 6.5: Detection capability when we send along multiple paths.

In addition to the sensitivity of the testing packets, we also consider two other parameters: the number of autonomous systems (AS) that a testing packet goes through, and the number of *bots* that send out the packets through disjoint paths. In Figure 6.5, each curve is labeled by a pair of numbers (a, b), in which a is the percentage that a packet is labeled as 'malicious' and is silently discarded by an AS, and b is the number of *bots* that send out the testing packets through disjoint paths.

From the figure we can learn the following patterns. (a) As the number of ASes increases, a testing packet has higher probability to be discarded by network devices along the path. The impacts are especially obvious for those highly sensitive packets. For example, the light blue line labeled by $(5, 1)$ suffers the most in detection accuracy when the number of ASes increases. (b) Using multiple *bots* to send out testing packets through different paths is an effective method to mitigate the impacts. For example, for highly sensitive packets (5% probability to be discarded), when we increase from

1 *bot* to 3 *bots*, the detection rate will jump from 77% to 99% even when the packets have to go through 5 ASes. (c) The packets with low or middle sensitivity levels are not greatly impacted by the path length. Based on these observations, t can choose *bot* accordingly based on the features of the testing traffic.

6.4.3 Future Extensions

In this part, we will discuss potential extensions to our approach. We are especially interested in the following aspects.

Incentive Model to Recruit *bot*

In our approach, the tester t who manages the third party test services and the *bots* who conduct the operations are two groups of nodes. Therefore, t must recruit nodes to serve as packet senders. To attract more nodes to the job, certain incentive must be provided. From a *bot*'s point of view, it trades its bandwidth and risks being identified as a 'potential network attacker' by its firewall or edge router for the profit that t can provide. Therefore, a corresponding cost-benefit model must be established so that a reasonable price can be charged to u and a fair split of the profit can be shared between t and *bot*.

Please note that this recruitment procedure is different from the crowd-sourcing problem [Li et al.(2019)Li, Li, Wang, and Wang, Wang et al.(2020)Wang, Tushar, Yuen, and Zhang] since the participants in crowd-sourcing usually contribute their labors but seldom risk their reputation. Some efforts such as [Moradi and Li(2020)] provide some hint from which we can benefit.

Reducing Overhead at t **and** *bot*

In our approach, the user u must digitally sign all the packets for test so that t and *bot* can verify the integrity and authenticity of the packets. Similarly, if a *bot* wants to prove that it strictly follows the procedure, it also needs to sign the packets with its private key. This operation, however, provides some hint for s to identify the testing

traffic since only a limited amount of traffic is digitally signed from end to end.

To solve this problem, we need to design a mechanism through which we can prove the integrity of t and *bot* without disclosing the objective of the packets. Some efforts from the automatic execution of smart contracts [Viglianisi et al.(2020)Viglianisi, Ceccato, and Tonella] could be our sources of solutions.

6.5 Conclusion

In this chapter we investigate the problem of proof of network security SLA. Specifically, a user could depend on a non-interest third party to assist it to test the enforcement of network SSLA. We first discuss the overview of the approach. The details of each step are then presented. We also focus on the authenticity and integrity of the test request, the execution of the test, and schemes to reduce false alarms. The quantitative results show that our approach can effectively detect the violation of the SSLA even when the user and tester do not have mutual trust. The user also has full control over the detection frequency and overhead.

When we put the research problem of the chapter in a bigger view, the goal is to allow end users to verify the execution of security service level agreement (SSLA) with the service providers. Different from the SLAs that focus on system performance, security related SLAs are hard to enforce since there is no obvious indicator of its execution. Our approach has the potential to be applied to other types of security services. These efforts will help end users to better protect their network and data security.

CHAPTER 7: Incentivisation of Outsourced Network Testing: View from Platform Perspective

7.1 Introduction

With fast development and wide adoption of Security as a Service (SaaS) [Hawedi et al.(2018)Hawedi, Talhi, and Boucheneb], more and more corporations start to depend on third party companies to protect their networks. This decision drastically reduces the workload of the CISO (Chief Information Security Officer) of the company since now she/he can focus on the policy level requirements and the expected security and robustness properties. However, the security services must be verified periodically to make sure that the service provider does not violate any of the service level agreements (SLA) [de Carvalho et al.(2017)de Carvalho, de Andrade, de Castro, Coutinho, and Agoulmine].

While the requirement on periodic verification sounds reasonable, it is quite hard to enforce in real life if the company depends on one or two nodes to conduct such test since their identities can be easily recognized and remembered by the service provider. In [Alasmari et al.(2020)Alasmari, Wang, and Wang], the authors propose an approach similar to the crowd-sensing system: a platform serves as the middleman to connect the customers who need their security properties to be tested and the nodes who can conduct such tests for them. The authors investigate the expected properties of the outsourced tests and the mechanisms to prevent either testers or customers from cheating.

While the approach in [Alasmari et al.(2020)Alasmari, Wang, and Wang] presents an overview of the platform, it lacks the discussion on an important problem: incentive models for the platform and network testers to participate in the verification

procedure. Note that to conduct network testing, a tester often needs to send out some packets that will be considered 'suspicious' or 'malicious' under some cases. For example, to assess whether or not the network security service provider will react promptly enough to identify some malicious payload, the tester may need to send out some packets that match to the malware signature. These packets, however, may be labeled by the tester's internet service provider (ISP) and causing negative consequence (e.g. limiting the network bandwidth of the tester). Therefore, a cost-benefit model must be established and analyzed before such a platform can be deployed.

To solve this problem, in this chapter we will establish a cost-benefit model from the platform point of view. We will first classify the network testing traffic based on the probability that they will be labeled and filtered by ISP. We will then establish a model for the problem as a linear programming problem. We analyze the complexity of the problem and design a heuristic algorithm to solve the incentive model under different situations. Finally, we apply the algorithm to multiple cost/benefit scenarios of outsourced network testing. The outputs of the algorithm explain the potential policies that the platform can adopt.

Our chapter has the unique contributions as follows. First, we establish an incentive model for outsourced network security testing. Different from traditional incentive models in which the participants trade resources for monetary incentives, in our model the testers try to maximize their profit while staying under the radar of network security monitors. Second, we model the problem as a linear programming problem and show that it is equivalent to variations of the knapsack problem. We then design a heuristic algorithm to solve the situation. Third, we apply our algorithm to multiple network testing scenarios and show that the outputs of the algorithm can explain the potential policies of the platform very well.

The remainder of the chapter is organized as follows. In Section 7.2, we describe related work that we can benefit from. In Section 7.3, we first discuss the differences

between our incentive problem and several problems that also demand incentive models. We then use a linear programming model to characterize the problem and show that it is equivalent to an NP hard problem. We design a heuristic algorithm to solve it. Section 7.4 presents the quantitative evaluation results and how the outputs match to the policy that a network testing platform can adopt. Finally, Section 7.5 concludes the chapter.

7.2 Related Work

In this part, we will describe the state-of-the-art research in several directions from which we can benefit. We are especially interested in the outsourced security services, their enforcement, and incentive models in other domains that we can refer to. For Security-as-a-Service (SaaS), researchers have conducted a lot of efforts to use Service Level Agreement (SLA) to define the criteria of evaluation. For example, the EU researchers built the framework SPECS [Rak et al.(2013)Rak, Suri, Luna, Petcu, Casola, and Villano] and the project MUSA [Rios et al.(2016)Rios, Mallouli, Rak, Casola, and Ortiz] that allowed users to prepare, negotiate, implement, and remediate security SLAs. The efforts in [Casola et al.(2020b)Casola, De Benedictis, Rak, and Villano] try to embed security into the system from the design phase. In [Boudi et al.(2019)Boudi, Farris, Bagaa, and Taleb], to alleviate the workload of security enforcement at the cloud service provider, authors try to distribute the efforts at edge nodes through lightweight virtualization. In [Hawedi et al.(2018)Hawedi, Talhi, and Boucheneb], the authors designed a different approach. They embed a lightweight IDS system at the cloud provider and allow the tenants to configure their own rules in the IDS for their VM. This approach provides a certain level of flexibility to users who have security expertise.

There are efforts in which the security provider allows end customers to participate in the configuration of security measures. For example, in [Casola et al.(2017)Casola, De Benedictis, EraÅcu, Modic, and Rak], end users can propose security ob-

jectives and the provider will determine and allocate resources to satisfy the needs. Example approaches are built to measure parameters for network security [Wonjiga et al.(2019b)Wonjiga, Rilling, and Morin] and data integrity [Wonjiga et al.(2019c)Wonjiga, Rilling, and Morin]. Note that the enforcement of the security measures still solely depends on the provider itself. A similar approach is to standardize the interfaces to the network security functions (NSF) in network function virtualization environments [Hyun et al.(2018)Hyun, Kim, Kim, Jeong, Hares, Dunbar, and Farrel]. In addition, software-defined networking can be imposed to optimize the security service process by implementing some of the packet filtering rules.

The definition and enforcement of security SLAs also benefit from the advances in the new techniques such as AI and smart contract. For example, in [Wonjiga et al.(2019c)Wonjiga, Rilling, and Morin], researchers applied blockchain to data integrity protection so that both end users and cloud providers can verify the results.

In the proposed approach, the platform will recruit a large number of testers to conduct the network security evaluation. During this procedure, the testers often need to send out data packets that are 'suspicious' or even sometimes labelled as 'malicious'. Therefore, we need to study how ISPs (internet service providers) are monitoring network traffic to assess the risk/cost of the testers. In [Haddadi et al.(2018)Haddadi, Christophides, Teixeira, Cho, Suzuki, and Perrig], the ISP compares the traffic patterns sampled from IoT devices with those at edge routers to detect anomaly traffic. A similar approach is applied to IDS at the wide area network level [Aqil et al.(2017)Aqil, Khalil, Atya, Papalexakis, Krishnamurthy, Jaeger, Ramakrishnan, Yu, and Swami]. In [Tedja et al.(2018)Tedja, Lim, and Ipung], the ISP uses the anomaly in DNS queries to detect command and control in botnets. ISPs have also used neural networks to analyze the URL access data to detect SQL injection attacks [Tang et al.(2020)Tang, Qiu, Huang, Lian, and Liu]. In [Tosun et al.(2021)Tosun, De Donno, Dragoni, and Fafoutis], the authors study the usage of residential IP addresses as proxies for mali-

cious network traffic, and propose to use the in and out traffic volume as a fingerprint to detect such activities.

Incentive models have been widely used to promote participation in the activities such as crowdsourcing [Islam et al.(2019)Islam, Alvi, Uddin, and Rahman, Muldoon et al.(2018)Muldoon, O'Grady, and O'Hare], crowdsensing [Khan et al.(2019)Khan, Ur Rehman, Zheng, Jan, and Alam, She(2020)], and next generation wireless networks [Luong et al.(2019)Luong, Wang, Niyato, Liang, Han, and Hou]. Compared to these models, our application scenario has some unique properties. Sending out 'suspicious traffic' to conduct network security tests may lead to identification and disconnection by ISP, thus leading to mid or long term negative impacts. The only scenarios we can find that share the properties are the financial incentives for clinical trials of treatment for infectious diseases [Kraft et al.(2019)Kraft, Duenas, Kublin, Shipman, Murphy, and Shah, Paul et al.(2021)Paul, Harbarth, Huttner, Thwaites, Theuretzbacher, Bonten, and Leibovici]. Here the volunteers will receive cash rewards for being contacting with infectious diseases and testing new medicines. They trade the short term incentives (cash here) for the potential of long term diseases (e.g. malaria). While the application scenarios are similar, the data from hospitals or medical systems only provide cash amount value but not the quantifiable incentive model.

7.3 Incentivisation of Outsourced Network Testing

In this chapter, we will present the details of the incentive model the platform can use to attract more nodes to participate in the network security services as testers. We will first discuss the assumed scenarios and the functionality of different parties in the system. We will then present the cost and payment from both the platform and the tester point of view. The incentivization model will then be formally defined as a linear programming problem with constraints. Our analysis will show that the problem is NP hard and heuristic approaches must be designed. We will also provide

a few such mechanisms.

7.3.1 System Assumptions

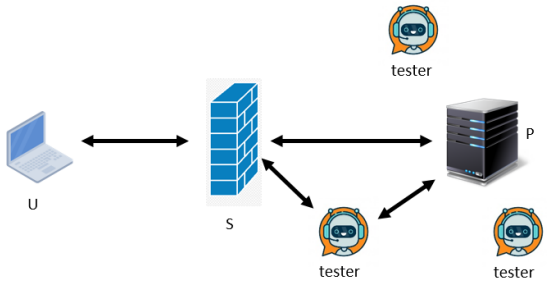


Figure 7.1: Application scenarios of the proposed approach.

Figure 7.1 shows the application scenario. We assume that an end user u uses the network security services provided by s. u wants to use a third party to verify whether or not s is satisfying the security service level agreement (SSLA). Therefore, it resolves to the network security service testing platform P. To simplify the scenario, we assume that u carefully crafts a group of packets and asks P to conduct the test. To prevent s from recognizing the source address of P and the testing traffic, P can recruit a group of *testers* to actually conduct the test. The test results can then be shared with u.

From the platform perspective, the overall operations must be profitable to make the business sustainable. For example, P will charge service fees from the user u and has to pay the testers correspondingly. Note that different testers may charge different fees to P because of the network service cost and security setup. While it is not necessarily true for P to make a profit on every request (e.g. temporary discount to attract users), in the long term it needs to make sure that ($income - $cost) is positive.

Different from the crowd-sensing or crowd-sourcing scenarios [Khan et al.(2019)Khan, Ur Rehman, Zheng, Jan, and Alam, Wei et al.(2020)Wei, Wu, and Long, Zhao et al.(2021)Zhao, Tang, Liu, and Zhang] in which the participants use valuable resources such as time and battery power to accomplish tasks,

the costs of network testers are usually caused by other factors. For example, should the testing machine is connected to a wall power outlet, battery lifetime is not a factor. The costs, however, are associated with the features of the testing traffic. For example, many ISPs will monitor the network traffic for potential attack detection [Aqil et al.(2017)Aqil, Khalil, Atya, Papalexakis, Krishnamurthy, Jaeger, Ramakrishnan, Yu, and Swami,Sacramento et al.(2018)Sacramento, Medeiros, Bota, and Correia]. Once a malicious or suspicious node is detected, the ISP may restrict its network bandwidth or sometimes directly disable the services. From this point of view, the testers are risking their network availability to participate in the services.

Because of the complexity of the ISP network monitoring and restriction policies, in this chapter we adopt the model presented in [Desai(2012)]. Here we classify the testing packets into three categories based on their sensitivity to security policies: low, middle, and high (our model could support more fine grained classification, which will be discussed in later sections). For each node i, within 24 hours of time period, the node can send out at most $R_{i,low}$, $R_{i,mid}$, and $R_{i,hi}$ packets at the low, medium, and high sensitivity levels, respectively, if it wants to avoid any restrictions by her ISP. Therefore, the platform P must consider such restriction when assigning tasks to the tester.

While we understand that the security of the proposed approach and the platform is essential for its success, in this chapter we focus on the incentive model, the complexity of the problem, and heuristic mechanisms. Some initial discussion on the security of the similar problem and its mitigation were presented in [Alasmari et al.(2020)Alasmari, Wang, and Wang]. We will continue to explore the security issues in future investigation.

The following table summarizes the symbols we use.

Table 7.1: Symbols used in the chapter.

P	platform that provides testing services
T_j	jth tester that conducts testing services
U_i	ith user that demands testing services
BP_h, BP_m, BP_l,	base price P charges for each testing packet at high, mid, and low sensitivity
$G_i(H_i, M_i, L_i)$	ith network testing request with H_i high, M_i middle, and L_i low sensitivity packets
$P_{j,l}$, $P_{j,m}$, and $P_{j,h}$	price tester j charges for sending a low, middle and high sensitivity packet
$R_{j,l}$, $R_{j,m}$, and $R_{j,h}$	the limits of low, middle, and high sensitivity packet tester j can send in 24hrs
$X_{i,j}$	whether or not we assign request G_i to tester j
$Y_{i,j,H}$	number of high sensitivity packets of request G_i assigned to tester T_j (similar for middle and low)
G	the set of network testing requests
J	the set of network testers

7.3.2 Working Procedure and the Cost Model

In this section, we will describe the working procedure of the platform and establish the cost model so that we can analyze the complexity of the overall problem.

Step 1: User u_i will submit a network testing request to platform P that consists of (H_{ui} high, M_{ui} middle, and L_{ui} low sensitivity packets. The user agrees to pay ($H_{ui} * BP_h + M_{ui} * BP_m + L_{ui} * BP_l$ to the platform;

Step 2: Platform P will assign the testing request to one or multiple testers. Note that different criteria and restrictions may need to be considered during this procedure. We will discuss two assignment methods in subsequent discussion. During this procedure, the platform needs to consider both its base price charge upon the user u_i and the price that it needs to pay to the tester to balance the book;

Step 3: The network testing will be conducted. Once the testing operations are accomplished, each party will pay the fee as the agreed amount. This request is complete.

7.3.2.1 Task Assignment Method 1

In this task assignment method, we assume that each testing request needs to be assigned to a single tester. In other words, the remaining packet limits of the tester must be large enough to hold the whole request. Therefore, we can formally define the problem as:

$$
\begin{aligned}
\text{maximize:} \quad & \sum_{i=1}^{|G|}\sum_{j=1}^{|J|} X_{i,j}(H_i * (BP_h - P_{j,h}) + \\
& M_i * (BP_m - P_{j,m}) + L_i * (BP_l - P_{j,l})) \\
\text{subject to:} \quad & \sum_{i=1}^{|G|} X_{i,j} * H_i \leq R_{j,h},\ X_{i,j} \in \{0,1\}, \quad j = 1\ to\ |J| \\
& \sum_{i=1}^{|G|} X_{i,j} * M_i \leq R_{j,m},\ X_{i,j} \in \{0,1\}, \quad j = 1\ to\ |J| \\
& \sum_{i=1}^{|G|} X_{i,j} * L_i \leq R_{j,l},\ X_{i,j} \in \{0,1\}, \quad j = 1\ to\ |J| \\
& \sum_{j=1}^{|J|} X_{i,j} = 1, \quad i = 1\ to\ |G|
\end{aligned}
$$

Here the objective is to maximize the profit of the platform by assigning the testing requests to testers. While P has uniform price rules for different types of testing traffic, each tester could charge different prices for the same type of packets since they need to consider the network costs. The profit P collects is determined by the difference between the two prices and the number of packets. At the same time, since a testing request cannot be decoupled and assigned to multiple testers, the total numbers of high, middle, and low sensitivity packets assigned to each tester must be within its limits, as shown in the constraints. Here $X_{i,j}$ represents whether or not

the requests G_i is assigned to tester j. The last constraint shows that each task is assigned to one tester.

Complexity Analysis

From the definition of the problem, we can see that it is related to the knapsack problem. Here each tester's packet limits are the knapsacks while the testing requests are the items. Below we will analyze the complexity of the problem.

Theorem 1: Assignment method 1 is an NP-complete problem.

Proof: We will show the equivalence between a special case of this problem and the subset knapsack problem. Let us assume that a polynomial algorithm A can solve the general problem described in assignment method 1. Now we consider a special case. Assume that each tester adopts the same price rules, thus we have $P_{j,l} = P_{j',l}$, $P_{j,m} = P_{j',m}$, $P_{j,h} = P_{j',h}$ for all different testers j and j'. Therefore, the profit that P makes through satisfying a network testing request will be the same whichever tester it assigns to. To further simplify the problem, we assume that for each tester j we have $R_{j,l}$ and $R_{j,m}$ as infinite. In other words, the only factor prevents a tester from accomplishing a request is its limit on high sensitivity packets. We also assume that $P_{j,l} = BP_l$ and $P_{j,m} = BP_m$. Now P can make a profit through only the high sensitivity packets.

Through the simplification, we have the following scenario: (1) each tester j has a knapsack with the capacity $R_{j,h}$; (2) the profit that P can make is proportional to the number of high sensitivity packets that it can fit into each knapsack; and (3) the goal of P is to maximize its profit.

This simplified problem is equivalent to the subset sum problem, which was classified as the Karp's 21 NP-complete problem [Karp(1972)]. ■

7.3.2.2 Task Assignment Method 2

In this task assignment method, we assume that a testing request can be broken down and assigned to multiple testers to jointly accomplish the task. Note that the whole request must be satisfied. In other words, we will not serve only a part of the request. Under this case, the assignment procedure could become more complicated since we can draw the packet capacity from different sources. As long as the remaining capacity of high, middle, and low sensitivity packets is enough for the request, we can satisfy the request.

From the first sight, it seems that this problem can be solved with a greedy algorithm, e.g., always assign the testing request to the tester that charges the lowest price for a category of the packets. The following example will show that this may not always be the optimal choice.

Here we will use a simplified example. Assume that all testers will adopt the same price table. Therefore, it does not matter to which tester we assign the task since the profit of the platform will be the same. We can put all the testing packet capacity together. Without losing generality, we assume that the testing packet capacity is (19(H), 19(M), 19(L)). Here all testers together can send out at most 19 test packets with high sensitivity, 19 with middle sensitivity, and 19 with low sensitivity. For each testing packet, the profit that the platform can collect is \$4 (H), \$2 (M), and \$1 (L), respectively. We also assume that the platform receives 4 requests, with the size of $R_1(10, 10, 10)$, $R_2(6, 6, 6)$, $R_3(5, 5, 5)$, $R_4(4, 4, 4)$, respectively. Here the numbers in parenthesis represent the high, middle, and low sensitivity packets in each task. Based on the table, the profit of each task is: for R_1, 10 * (\$4 + \$2 + \$1) = \$70, for R_2, 6 * (\$4 + \$2 + \$1) = \$42, for R_3, \$35, and for R_4, \$28.

If a greedy assignment algorithm is adopted, we will first accomplish task R_1 and R_2. At this time, the remaining testing packet capacity is not enough for any other task, so the platform's profit will be \$112. However, we know that the platform can

actually satisfy tasks R_1, R_3, and R_4 to make the profit \$133.

Below we will formalize the assignment problem as follows:

$$\begin{aligned}
\text{maximize:} \quad & \sum_{i=1}^{|G|}\sum_{j=1}^{|J|}(Y_{i,j,H} * (BP_h - P_{j,h}) + Y_{i,j,M} * \\
& (BP_m - P_{j,m}) + Y_{i,j,L} * (BP_l - P_{j,l})) \\
\text{subject to:} \quad & \sum_{i=1}^{|G|} Y_{i,j,H} \leq R_{j,h}, \quad j = 1 \; to \; |J| \\
& \sum_{i=1}^{|G|} Y_{i,j,M} \leq R_{j,m}, \quad j = 1 \; to \; |J| \\
& \sum_{i=1}^{|G|} Y_{i,j,L} \leq R_{j,l}, \quad j = 1 \; to \; |J| \\
& \sum_{j=1}^{|J|} Y_{i,j,H} = H_i, \quad i = 1 \; to \; |G| \\
& \sum_{j=1}^{|J|} Y_{i,j,M} = M_i, \quad i = 1 \; to \; |G| \\
& \sum_{j=1}^{|J|} Y_{i,j,L} = L_i, \quad i = 1 \; to \; |G|
\end{aligned}$$

Here we can assign a request to multiple testers as long as all of the testing packets are covered. $Y_{i,j,H}$ represents the number of high sensitivity packets of request G_i that are assigned to tester T_j. Another restriction is the total number of high (middle, or low) sensitivity packets that are assigned to a tester is within her limit.

Complexity Analysis

Since now a single testing request can be assigned to multiple testers, it is different from the scenario that we discussed in Section 3.2.1. Since the testing requests can

be assigned to multiple testers, the testing capacity of different nodes can actually be merged together to form a large pool, as shown in Figure 7.2. Please note that since each tester may have different capacities in high, middle, and low sensitivity packets, their sums are also different. At the same time, each tester has its own price policy of the packets.

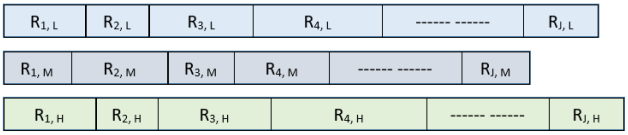


Figure 7.2: Testing capacity forms a large pool.

Since the testing capacity can be merged to form a large pool, we have a variant of the multi-dimensional knapsack problem [Akcay et al.(2007)Akcay, Li, and Xu, Laabadi et al.(2018)Laabadi, Naimi, El Amri, and Achchab]. Here the profit of satisfying a single testing request is not a constant since each tester has its own price policy. Below we will prove that this variant is still an NP-hard problem.

Theorem 2: Assignment method 2 is an NP-hard problem.

Proof: We will show the equivalence between a special case of this problem and the general multidimensional knapsack problem. Let us assume that a polynomial algorithm B can solve the general request assignment problem described in assignment method 2. Now we consider a special case. Assume that each tester adopts the same price rules, thus we have $P_{j,l} = P_{j',l}$, $P_{j,m} = P_{j',m}$, $P_{j,h} = P_{j',h}$ for all different testers j and j'. Therefore, the profit that P makes through satisfying a network testing request will be the same whichever tester it assigns to. At the same time, note that we have put all the testing capacity together to form a large, 3-dimensional (high, middle, and low) knapsack. Given a set of testing requests, the algorithm B will solve the profit maximization problem in Method 2 in polynomial time.

Through this simplification, we have created a 0-1 MDKP (multi-dimensional knapsack problem) with the dimension number of 3. Based on [Garey and Johnson(1979), Lin(1998)], the 0-1 MDKP problem is strongly NP-Hard. Therefore, if the

algorithm B exists, we will have a polynomial solution to the MDKP problem.

■

7.3.3 Heuristic Algorithms

Since the request assignment problems cannot be solved within polynomial time, in this section we will discuss some heuristic algorithms. Depending on whether or not a testing request can be assigned to multiple testers, we discuss two different approaches.

The first heuristic algorithm we present is a variation of the Primal Effective Capacity Heuristic (PECH) mechanism for the general MDKP [Akcay et al.(2007)Akcay, Li, and Xu]. Specifically, it is a greedy algorithm for the Task Assignment Method 1. Here we assume that a task must be assigned to a single tester. Since each tester adopts its own price model, the profit that P can collect from accomplishing the task depends on the assigned tester. Figure 7.3 shows the pseudo code of the algorithm. While the overall procedure is straight forward, the operation in step (4) could be conducted in different ways. Here the platform needs to select one task from all the unassigned network testing requests. The selection method could directly impact the final result. Several examples of the selection criteria include: (1) first come first serve; (2) fit as many as possible requests to the testing capacity (thus P inclines to choose the tasks with fewer packets); or (3) randomly choose a task.

Another criteria of tester assignment is the traffic capacity usage of different categories. Since we assign a whole request to one tester, we try to use the packet capacities in a balance way to avoid the situations in which a certain type of packet capacity is used up and while for other types a plenty of capacity is reamined. Under this case, we will assign a request to a tester that will create the least imbalance in its capacities after satisfying the request. Below we provide an example. Assume that we have two testers t_1 and t_2 who have used their capacities from high to low as follows: $t_1(71\%, 68\%, 72\%)$ and $t_2(66\%, 69\%, 67\%)$. So the imbalance of t_1 is $72\% - 68\% = 4\%$,

and for t_2 is 3%. Now assume that a request contains only high sensitivity packets. Because of the difference in capacities of testers, it will use 2% of t_1's capacity or 3% of t_2's capacity. Therefore, if we assign it to t_1, the new imbalance value will be ((71% + 2%) - 68% = 5%). While for t_2 the new value is (69% - 67% = 2%). Therefore, to reduce imbalance at testers, we will give the task to t_2.

Instead of maximizing the profit from the current request, we could adopt other criteria during the procedure to select the testers. For example, the platform needs to attract and maintain a large number of testers to keep the service sustainable. Therefore, it needs to assign tasks to different testers even if that means loss of profit. If that is the case, we can choose the tester who has the largest percentage of unused testing capacity to achieve load balance.

```
Algorithm 1. Greedy task assignment algorithm
Begin
    Initialize the capacity of each tester;                                  (1)
    Initialize the set of tasks G;                                           (2)
    While (G is not empty)                                                   (3)
    {
        Select task Gi for assignment;                                       (4)
        Find the set T of testers whose remaining capacity is enough         (5)
            to satisfy Gi;
        Remove Gi from G;                                                    (6)
        Choose the tester Tj from T that will make the largest profit for    (7)
            satisfying Gi;
        Assign Gi to Tj and update Tj's capacity;                            (8)
    }
End
```

Figure 7.3: Greedy assignment algorithm when each task must be assigned to a single tester.

The Task Assignment Method 2 is a little bit different since we can assign the packets to multiple testers. Therefore, a greedy algorithm will try to assign each single testing packet to the tester who charges the lowest price. Once that tester's capacity is reached, we can move on to the next cheapest tester. Note that this approach tries to maximize the profit of the current request for the platform. If the first-come-first-serve method is always adopted, it is possible that a certain type of packet capacity is used up first, thus preventing us from admitting new requests. For

example, if all testing capacity of the middle level sensitivity packets is used up, we will not be able to admit any request that contains middle sensitivity packets since we do not allow a request to be partially satisfied.

To prevent this scenario from happening, we can manage the remaining capacity of different types of packets and try to maintain a balance. For example, during the request assignment we can set up a threshold that after admitting any testing request, the difference in remaining capacity of different types of packets in percentage will not exceed this value. Below we provide an example. Assume that we set the threshold at 5%. Before admitting a request, the capacity usage are 45% (low), 42% (middle), and 47% (high), respectively. Now if a task requests 1% of middle sensitivity packet capacity and 2% of high, we will not admit it since the ending capacity usage will be 43% (middle) and 49% (high) which will be larger than the threshold 5%. Note that this method tries to improve the capacity usage to maximize the profit.

7.4 Quantitative Results

In this part, we will present some quantitative results about the proposed approaches. The experiments focus on the achieved profit of different approaches, and the practicability of the task assignment models.

7.4.1 Achievable Profit vs Heuristic Approaches

Based on the discussion in previous sessions, we can see that the task assignment problem is an NP problem. Therefore, in this section, we will compare the maximum profit under some scenarios to the achievable profit of the heuristic approaches. Restricted by the search space size and required computation power, we will experiment with some small scale questions.

We assume that the prices that the platform charges for high, middle, and low sensitivity packets are \$12, \$10, and \$8, respectively. For each tester, the capacities of high, middle, and low sensitivity packets follow uniform distribution in the

ranges (900, 1100), (1800, 2200), (900, 1100), respectively. The size of the network test requests also follows uniform distribution around the expected values. They are divided into two groups. The first group have the sizes that range from 20% to 90% of the testers' capacities, while the second group range from 5% to 45%. The charging prices of the testers uniformly distribute between 95% to 100% of the platform prices. To calculate the maximum profit of the assignment, we search for all possible combinations.

Table 7.2: Maximum profit vs heuristic approaches.

# of testers	# of Grp 1 request	Group 1 req size	# of Grp 2 request	Group 2 req size	Maximum Profit ($)	FCFS Profit ($)	Random Profit ($)
1	8	90%	8	45%	1525.34	1466.66	1477.82
1	8	80%	8	40%	1393.38	1312.86	1319.06
1	8	70%	8	35%	1229.38	1148.86	1155.06
1	8	60%	8	30%	1555.3	1519.42	1525.62
1	8	50%	8	25%	1603.7	1385.42	1440.06
1	8	40%	8	20%	1613.46	1354.24	1388.02
1	8	30%	8	15%	1577.2	1400.88	1413.82
1	8	20%	8	10%	1584.76	1421.54	1470.4
2	8	90%	8	45%	3005.98	2929.48	2940.32
2	8	80%	8	40%	2688.22	2605.48	2616.32
2	8	70%	8	35%	2364.22	2281.48	2292.32
2	8	60%	8	30%	3022.18	2919.24	2947.64
2	8	50%	8	25%	2976.22	2617.88	2646.58
2	8	40%	8	20%	3005.34	2571.4	2634.2
2	8	30%	8	15%	3015.26	2820.78	2842.82

In this group of experiments, we consider three task assignment mechanisms: first come first serve (FCFS), random assignment, and exclusive search (maximum profit). Here the FCFS mechanism will try to satisfy the tasks based on their arriving order. The random assignment mechanism picks from the pool of unsatisfied tasks and assign it to the tester that will generate the highest profit. In Figure 7.4, we show the ratio between the profits of the mechanisms and the maximum profit.

From the figure, we can see that the size of the network test requests has an

large impact on the achievable profit. For example, when the sizes of the requests are comparable to the capacities of the testers (60% to 90%), very frequently we can fit only one request into the tester's capacity. Therefore, the difference between the maximum profit and the achievable profits of the assignment mechanisms is not large. When the size of the requests decreases (40% and 50%), we can actually assign multiple requests to a tester. Therefore, the selection of testers could impact the profit a lot since a good assignment mechanism can often use higher percentage of the capacity, thus causing larger differences between the maximum profit and achievable profits of the task assignment methods. When the size of the requests further decreases, we can fit many requests into the capacity of a single tester, and the percentage usage goes up again. From this point of view, after the platform learns the distribution of the request sizes, they can recruit testers that can help the platform to increase its profit. From another aspect, we can see that the FCFS and Random assignment mechanisms demonstrate similar performance. The reason is that random assignment can be viewed as another case of FCFS. The only difference is the order of received requests. However, when the requests follow the same distribution and the number of requests is large enough, their performance will be similar.

Figure 7.4: Relationship between the maximum profit and profit of different mechanisms.

7.4.2 Comparison of Heuristic Approaches

In the second group of experiments, we will compare multiple heuristic approaches. Specifically, we study 4 heuristic mechanisms: (1) FCFS/Greedy: in this mechanism, the platform adopts the first-come-first-serve policy. A single request must be assigned

to one tester who can generate the largest profit through accomplishing the request. (2) Random/Greedy: in this method, we assume that all requests are available from the beginning. We will then randomly pick requests from the pool and assign it to a tester who can generate the largest profit. From this point of view, it is similar to FCFS but we choose requests randomly instead of based on the arriving order.

The third mechanism that we experiment with is called "balanced capacity usage". In this method, we try to assign a request to a tester to minimize the difference between the used capacity in high, middle, and low sensitivity packets at the tester. Specifically, for all tester T_j, we try to min ($max\ capacity\ usage\ difference$). The objective is to use the capacities in a balanced way so that we can assign more requests to a single tester.

Last but not least, in the 'merged' mechanism, we allow a request to be assigned to multiple testers and each accomplishes only a part of the task. In this method, the capacities of the high, middle, and low sensitivity packets are actually merged into one pool. For each request, we will choose the tester who charges the lowest price to the corresponding packet type. Since different testers may charge different prices, we may assign the high sensitivity packet to one tester and the middle sensitivity packets to another tester. In this way, we can achieve higher profit since we will use the tester capacities more effectively.

Since in this group of experiments we will not search for the optimum solution, we can simulate a scenario with much more testers and requests. We assume that the platform recruits 50 testers and there are 1500 requests submitted by end users. The meanings of the parameters are similar to those discussed in Section 7.4.1. Since we do not have the maximum profit value, in the following figures we will post the absolute profit (in $).

Figure 7.5 shows the quantitative results. On the X-axis, we have the ratio between the size of the network testing requests and the tester capacity. On the Y-axis, we have

the profit value. First, for the 'merged' mechanism, since we put all the capacities of the testers into one pool, we can continue to satisfy the requests until at least one of the categories can no longer fit any request (e.g. the smallest low sensitivity demand of the remaining requests is 50 while the remaining tester capacity in that category is 30). Therefore, we can see that the profit of the 'merged' mechanism is not largely impacted by the request size. On the contrary, for the other three mechanisms, their profit will slowly decrease as the size of the requests increases.

Compared to 'FCFS/Greedy' and 'Random/Greedy', the 'Balanced' mechanism tries to use the capacity in different categories in a strategic manner so that we can avoid the scenario that one category is exhausted first while a large portion of capacities of other categories is still unused. From the figure, we can see that the 'Balanced' mechanism generates higher profit than the other two methods. As the size of the requests increases, more capacity at the testers could not be used. Therefore, the overall profit will decrease. Similar to the reason we discussed in Section 7.4.1, the 'FCFS/Greedy' and 'Random/Greedy' mechanisms do not demonstrate noticeable differences.

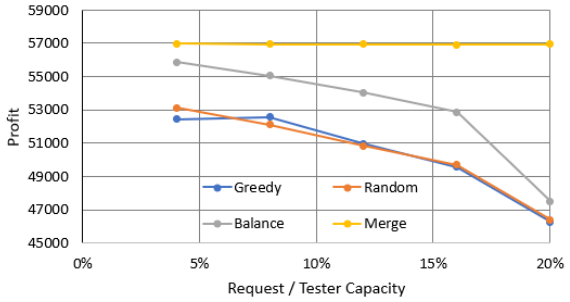


Figure 7.5: Detection capability when we send along multiple paths.

In Figure 7.6, we study how the price model of the testers impacts the profit of the platform. Specifically, we adjust the price difference between the testers and the platform. In the X-direction, the average price difference increases from 2% to about 10%. On the Y-direction we show the profit. From the figure we can see that as the price difference changes, the profit increases almost linearly.

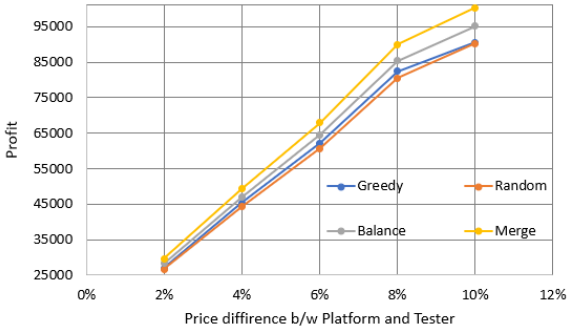


Figure 7.6: Detection capability when we send along multiple paths.

7.4.3 Future Extensions

In this part, we will discuss potential extensions to our approach. We are especially interested in the following aspects.

Impacts of Price Model

This chapter can be viewed as our exploration of the practicability of the outsourced network testing services. Specifically, we want to see how various factors impact the profit of the platform. To simplify the discussion and simulation, we assume a uniform distribution of the price difference between the testers and platform. In real life, both platform and testers could adopt more complicated price model. For example, they can dynamically adjust the price based on the number of requests, the number of testers, and the remaining capacity. From this point of view, maximization of the platform profit will become a more challenging question.

Long Term/Short Term Impacts on Testers

In this chapter, we assume an on/off model for the impacts on testers: as long as a tester sends out the testing packets below the threshold, there will be no negative impacts on their network usage. In real life, the impacts model will be more complicated. For example, an ISP may restrict the tester's network bandwidth based on the number of suspicious packets that it sends out in the last week. Therefore, the operation of a tester will have a long term impact on its capability to participate in subsequent network testing operations (and generate profit).

Note that this problem is different from the recruitment problem in traditional crowd-sourcing domains [Li et al.(2019)Li, Li, Wang, and Wang, Wang et al.(2020)Wang, Tushar, Yuen, and Zhang] since most cost/benefit models do not have long term impacts. This problem is similar to the cases in which incentives are offered by pharmaceutical companies to volunteers to test their new medicines since the intentional infection of a disease may have long term negative impacts on the patient that are not known at the test time. We could benefit from the previous research in this domain [Hoffart and Scheibehenne(2019), Largent and Lynch(2017)].

Privacy of Security Policies

Another concern of the outsourced network testing is the leakage of the network security policies of the end customers. Based on the construction and contents of the network testing packets, a network tester may derive out the network security policies that the end user adopts in its system. Therefore, should a tester turn malicious, it has the capability to design attacks based on the knowledge of the policy. To defend against such attacks, a potential mechanism is to distribute the testing traffic to multiple testers so that each party will hold only a portion of the knowledge. The end user could also embed deceitful traffic into the request (with increased cost). Algorithms need to be designed to assign testing packets to different parties.

7.5 Conclusion

In this chapter we investigate the problem of outsourced network test. Specifically, we focus on the working model of the platform and the possible mechanisms of task assignment. Our analysis shows that the task assignment problem is an NP problem and we need to design heuristic algorithms to assist the platform to generate higher profit. We conduct simulation to investigate the maximum profit and the achievable profits of different approaches. Our simulation shows that maintaining a balance between the remaining testing capacities of different categories will increase profit of the platform.

When we put the research problem of the chapter in a bigger view, the goal is to allow the platform to run the network testing services in a sustainable way. Therefore, it needs to attract enough number of end users as well as network testers to the platform. At the same time, it needs to protect the privacy of the end users. Different from the service level agreements that focus on resource usage such as CPU cycles, outsourcing of security related SLAs deserves more careful execution since a balance between the safety and efficiency must be maintained. We will investigate more realistic price model and network usage models in future studies.

CHAPTER 8: Conclusions

This thesis explored and devised several user based SSLA security enforcement approaches to detect SSLA violations. We demonstrated that it is feasible for the user to verify cloud security services without requiring high computation overhead or causing adverse availability impact that cause disruption to the service provider infrastructure. First, we presented the Proof Of Encryption (PoE) problem and devised the security audit properties that are needed to achieve efficient audit approach where computation cost and overhead is as low as possible. Then, we proposed two security approaches to enable users to verify the encryption algorithm and key strength are honored by the service provider. The experiment results are applied to evaluate the proposed security mechanisms.

Second, the end user has the option to choose homomorphic encryption algorithm for enforcement, the proposed security security approach is evaluated on several metrics such as detection capability , resilience to false positive results and incurred overhead. In order to verify the security approaches, we developed a challenge and verification procedure, then the security protocol is evaluated for correctness using BAN logic.

The third security verification approach focused on studying outsourced network security scanning services. The user selects a third party auditor (TPA) to execute the audit test using crafted network traffic data that are securely shared with the tester. The tester can simply conduct the security verification using network packets, and sends back the testing results to the user to detect if any SSLA violation occured.

In the fourth contribution, an incentive model is proposed to ensure that network security testers are incentivized and to maximize the overall platform profit. The

proposed approach proves that the task assignment problem is an NP problem and we designed heuristic algorithms to assist the platform to generate higher profit. We conducted an experiment to investigate the maximum profit and the achievable profits of different approaches. Our simulation shows that maintaining a balance between the remaining testing capacities of different categories will increase profit of the platform.

References

Akcay Y, Li H, Xu S (2007) Greedy algorithm for the general multidimensional knapsack problem. In: Annals of Operations Research, vol 150, pp 17–29

Akter S, Whaiduzzaman M (2017) Dynamic service level agreement verification in cloud computing. International Journal of Computer Science and Information Security (IJCSIS) 15(9)

Al Shebli H, Beheshti B (2018) A study on penetration testing process and tools. In: IEEE Long Island Systems, Applications and Technology Conference (LISAT), pp 1–7

Alasmari S, Wang W, Wang Y (2020) Proof of network security services: Enforcement of security sla through outsourced network testing. In: International Conference on Communication and Network Security (ICCNS), pp 52–59

Alliance CS (2019) Top threats to cloud computing the egregious. https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven/

Anderson RJ (2008) Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley

Andrieux A, Czajkowski K, Dan A, Keahey K, Ludwig H, Nakata T, Pruyne J, Rofrano J, Tuecke S, Xu M (2007) Web services agreement specification (ws-agreement). https://www.ogf.org/documents/GFD.107.pdf

Aqil A, Khalil K, Atya AO, Papalexakis EE, Krishnamurthy SV, Jaeger T, Ramakrishnan KK, Yu P, Swami A (2017) Jaal: Towards network intrusion detection at isp scale. In: Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies, pp 134â–146

Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson Z, Song D (2007) Provable data possession at untrusted stores. In: Proc. of CCS, pp 598–â609

Bhasker B, Murali S (2020) A survey on security issues in sensor cloud environment for

agriculture irrigation management system. Journal of Critical Reviews 7(4):1–10

Boudi A, Farris I, Bagaa M, Taleb T (2019) Assessing lightweight virtualization for security-as-a-service at the network edge. IEICE TRANSACTIONS on Communications E102-B(5):970–977

Bresson E, Chevassut O, Pointcheval D, Quisquater JJ (2001) Provably authenticated group diffie-hellman key exchange. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp 255–264

de Carvalho C, de Andrade R, de Castro M, Coutinho E, Agoulmine N (2017) State of the art and challenges of security sla for cloud computing. Computers Electrical Engineering 59:141–152

Casola V, De Benedictis A, Rak M, Villano U (2015) Sla-based secure cloud application development: The specs framework. In: 2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, pp 337–344

Casola V, De Benedictis A, Rak M, Rios E (2016) Security-by-design in clouds: A security-sla driven methodology to build secure cloud applications. Procedia Computer Science 97:53–62

Casola V, De Benedictis A, EraÅcu M, Modic J, Rak M (2017) Automatically enforcing security slas in the cloud. IEEE Transactions on Services Computing 10(5):741–755

Casola V, Benedictis AD, Rak M, Villano U (2020a) A methodology for automated penetration testing of cloud applications. International Journal of Grid and Utility Computing 11(2)

Casola V, De Benedictis A, Rak M, Villano U (2020b) A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach. Journal of Systems and Software 163:110537

Chaudhary P, Gupta R, Singh A, Majumder P (2019) Analysis and comparison of

various fully homomorphic encryption techniques. In: International Conference on Computing, Power and Communication Technologies (GUCON), pp 58–62

Chen B, Wu X, Lu W, Ren H (2019) Reversible data hiding in encrypted images with additive and multiplicative public-key homomorphism. Signal Processing 164:48–57

CloudTrust Protocol Working Group (2015) Cloudtrust protocol data model and api. Cloud Security Alliance

Control CCS (2022) Cloud computing compliance controls catalogue (c5). https://www.bsi.bund.de/SharedDocs/Downloads/BSI/Publications/CloudComputing/

Dastjerdi AV, Tabatabaei SGH, Buyya R (2012) A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. Software: Practice and Experience 42(4):501–518

Denis M, Zena C, Hayajneh T (2016) Penetration testing: Concepts, attack methods, and defense strategies. In: IEEE Long Island Systems, Applications and Technology Conference (LISAT), pp 1–6

Desai VR (2012) Techniques for detection of malicious packet drops in networks. Master's thesis, University of Massachusetts Amherst

EC Cloud Select Industry Group (C-SIG) (2014) Cloud service level agreement standardization guidelines. European Commission

Emeakaroha VC, Brandic I, Maurer M, Dustdar S (2010) Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In: 2010 International Conference on High Performance Computing & Simulation, IEEE, pp 48–54

Erkuden Rios and Eider Iturbe and Xabier Larrucea and Massimiliano Rak and etc (2019) Service level agreement-based gdpr compliance and security assurance in (multi)cloud-based systems. IET Software

FISMA (2022) Open web application security project. https://www.cisa.gov/federal-

information-security-modernization-act

Foerster K, Schmid S, Vissicchio S (2019) Survey of consistent software-defined network updates. IEEE Communications Surveys Tutorials 21(2):1435–1461

Fun TS, Samsudin A (2016) A survey of homomorphic encryption for outsourced big data computation. KSII Transactions on Internet and Information Systems 10(8):3826–3851

Gadepally V, Hancock B, BKaiser, Kepner J, PMichaleas, Varia M, Yerukhimovich A (2015) Computing on masked data to improve the security of big data. In: Proc. of IEEE Symposium HST, pp 1–6

Gao T, Deng X, Wang Y, Kong X (2018) Paas: Pmipv6 access authentication scheme based on identity-based signature in vanets. IEEE Access 6:37480–37492

Garey MR, Johnson DS (1979) Computers and Intractability : A Guide to the Theory of NP-Completeness. W. H. Freeman, San Francisco

Gartner (2019) Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019

Giachino E, de Gouw S, Laneve C, Nobakht B (2016) Statically and dynamically verifiable sla metrics. In: Abraham E, Bonsangue M, Johnsen E (eds) Theory and Practice of Formal Methods. Lecture Notes in Computer Science, vol 9660, Springer, Cham

Gope P, Sikdar B (2019) Lightweight and privacy-preserving two-factor authentication scheme for iot devices. IEEE Internet of Things Journal 6(1):580–589

Griffo C, Almeida JPA, Guizzardi G, Nardi JC (2019) Service contract modeling in enterprise architecture: An ontology-based approach. Information Systems p 101454, https://doi.org/10.1016/j.is.2019.101454, URL `http://www.sciencedirect.com/science/article/pii/S030643791930506X`

Gueron S (2012) Intel advanced encryption standard (intel aes) new instructions set. Intel Whitepaper, 323641-001

Haddadi H, Christophides V, Teixeira R, Cho K, Suzuki S, Perrig A (2018) Siotome: An edge-isp collaborative architecture for iot security. In: Proceedings of International Workshop on Security and Privacy for the Internet-of-Things (IoTSec)

Hale ML, Gamble R (2013) Building a compliance vocabulary to embed security controls in cloud slas. In: 2013 IEEE Ninth World Congress on Services, IEEE, pp 118–125

Hao K, Xin J, Wang Z, Jiang Z, Wang G (2018) Decentralized data integrity verification model in untrusted environment. In: Asia-Pacific Web (APWeb) and Web-age information management (WAIM) joint international conference on Web and big data, Springer, pp 410–424

Hawedi M, Talhi C, Boucheneb H (2018) Security as a service for public cloud tenants(saas). Procedia Computer Science 130:1025–1030

Hermanto BR, Iskandar, Hendrawan, Edward IJM (2019) Implementation of service level measurement based on system uptime sensor of network device in internet connection service. In: IEEE International Conference on Wireless and Telematics (ICWT), pp 1–4

HIPPA (2015) Encryption almost prevents humana data breach in wisconsin. HIPAA Journal

Hoffart J, Scheibehenne B (2019) Pill or bill? influence of monetary incentives on the perceived riskiness and the ethical approval of clinical trials. Judgment and Decision Making 14(2):130–134

Hui H, Zhou C, An X, Lin F (2019) A new resource allocation mechanism for security of mobile edge computing system. IEEE Access 7:116886–116899

Hussain M, Al-Mourad MB (2014) Effective third party auditing in cloud computing. In: International Conference on Advanced Information Networking and Applications Workshops, pp 91–95

Hyun S, Kim J, Kim H, Jeong J, Hares S, Dunbar L, Farrel A (2018) Interface to net-

work security functions for cloud-based security services. IEEE Communications Magazine 56(1):171–178

Islam L, Alvi ST, Uddin MN, Rahman M (2019) Obstacles of mobile crowdsourcing: A survey. In: IEEE Pune Section International Conference (PuneCon), pp 1–4

Juels A, Kaliski BS (2007) Pors: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp 584–â597

Kaaniche N, Mohamed M, Laurent M, Ludwig H (2017) Security sla based monitoring in clouds. In: IEEE International Conference on Edge Computing (EDGE), pp 90–97

Kanich C, Kreibich C, Levchenko K, Enright B, Voelker GM, Paxson V, Savage S (2008) Spamalytics: An empirical analysis of spam marketing conversion. In: Proceedings of the ACM Conference on Computer and Communications Security, pp 3â–14

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, Bohlinger JD (eds) Complexity of Computer Computations: The IBM Research Symposia Series, Springer, Boston

Khan F, Ur Rehman A, Zheng J, Jan MA, Alam M (2019) Mobile crowdsensing: A survey on privacy-preservation, task management, assignment models, and incentives mechanisms. Future Generation Computer Systems 100:456–472

Khettab Y, Bagaa M, Dutra DLC, Taleb T, Toumi N (2018) Virtual security as a service for 5g verticals. In: IEEE Wireless Communications and Networking Conference (WCNC), pp 1–6

Kim D, Jeong IR (2017) Certificateless public auditing protocol with constant verification time. Security and Communication Networks 2017

Kouki Y, Ledoux T (2012) CSLA: a language for improving cloud SLA management. In: International Conference on Cloud Computing and Services Science

(CLOSER), pp 586â–591

Kraft SA, Duenas DM, Kublin JG, Shipman KJ, Murphy SC, Shah SK (2019) Exploring ethical concerns about human challenge studies: a qualitative study of controlled human malaria infection study participantsâ motivations and attitudes. Journal of Empirical Research on Human Research Ethics 14(1):49–60

Krotsiani M, Kloukinas C, Spanoudakis G (2017) Validation of service level agreements using probabilistic model checking. In: 2017 IEEE International Conference on Services Computing (SCC), IEEE, pp 148–155

Laabadi S, Naimi M, El Amri H, Achchab B (2018) The 0/1 multidimensional knapsack problem and its variants: A survey of practical models and heuristic approaches. American Journal of Operations Research (8):395–439

Largent E, Lynch H (2017) Paying research participants: The outsized influence of "undue influence". IRB 39(4):1â–9

Lee C, Kavi KM, Paul RA, Gomathisankaran M (2015) Ontology of secure service level agreement. In: IEEE International Symposium on High Assurance Systems Engineering, pp 166–172

Li H, Li T, Wang W, Wang Y (2019) Dynamic participant selection for large-scale mobile crowd sensing. IEEE Transactions on Mobile Computing 18(12):2842–2855

Li Y, Yu Y, Yang B, Min G, Wu H (2018) Privacy preserving cloud data auditing with efficient key update. Future Generation Computer Systems 78:789– 798

Lin EYH (1998) A bibliographical survey on some well-known non-standard knapsack problems. INFOR 36(4):274–â317

Liu X, Xia C, Wang T, Zhong L, Li X (2020) A behavior-aware sla-based framework for guaranteeing the security conformance of cloud service. Frontiers of Computer Science 14:1–17

Lu N, Zhang Y, Shi W, Kumari S, Choo KKR (2020) A secure and scalable data

integrity auditing scheme based on hyperledger fabric. Computers & Security 92:101741

Luna J, Suri N, Iorga M, Karmel A (2015) Leveraging the potential of cloud security service-level agreements through standards. IEEE Cloud Computing 2(3):32–40

Luong NC, Wang P, Niyato D, Liang YC, Han Z, Hou F (2019) Applications of economic and pricing models for resource management in 5g wireless networks: A survey. IEEE Communications Surveys and Tutorials 21(4):3298–3339

Madi T, Majumdar S, Wang Y, Jarraya Y, Pourzandi M, Wang L (2016) Auditing security compliance of the virtualized infrastructure in the cloud: Application to openstack. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp 195–206

Madi T, Jarraya Y, Alimohammadifar A, Majumdar S, Wang Y, Pourzandi M, Wang L, Debbabi M (2018) Isotop: auditing virtual networks isolation across cloud layers in openstack. ACM Transactions on Privacy and Security (TOPS) 22(1):1–35

Majumdar S, Madi T, Wang Y, Jarraya Y, Pourzandi M, Wang L, Debbabi M (2015) Security compliance auditing of identity and access management in the cloud: Application to openstack. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp 58–65

Majumdar S, Tabiban A, Jarraya Y, Oqaily M, Alimohammadifar A, Pourzandi M, Wang L, Debbabi M (2019) Learning probabilistic dependencies among events for proactive security auditing in clouds. Journal of Computer Security 27(2):165–202

Moradi M, Li Q (2020) Rogue people: on adversarial crowdsourcing in the context of cyber security. Journal of Information, Communication and Ethics in Society ahead-of-print(ahead-of-print)

Muldoon C, O'Grady MJ, O'Hare GM (2018) A survey of incentive engineering for

crowdsourcing. Knowledge Eng Review 33:e2

MUSA (2015) Multi-cloud secure applications. http://www.musa-project.eu.

Nawaz F, Janjua NK, Hussain OK, Hussain FK, Chang E, Saberi M (2018) Event-driven approach for predictive and proactive management of sla violations in the cloud of things. Future Generation Computer Systems 84:78 – 97, https://doi.org/10.1016/j.future.2018.02.025, URL http://www.sciencedirect.com/science/article/pii/S0167739X1732280X

NIST2 (2015) A profile for u. s. federal cryptographic key management systems. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-152.pdf2

OWASP2 (2022) Open web application security project. https://owasp.org/

Paul M, Harbarth S, Huttner A, Thwaites G, Theuretzbacher U, Bonten M, Leibovici L (2021) Investigator-initiated randomized controlled trials in infectious diseases: Better value for money for registration trials of new antimicrobials. Clinical Infectious Diseases 72(7):1259–â1264

PCI-DSS (2022) Payment card industry. https://www.pcisecuritystandards.org/pcisecurity/

Popa R, Redfield C (2011) Cryptdb: Protecting confidentiality with encrypted query processing. In: Proc. of ACM SOSP, pp 85–100

Pornin T (2018) Bearssl: A smaller ssl/tls library. https://bearssl.org

Pourpouneh M, Ramezanian R (2016) A short introduction to two approaches in formal verification of security protocols: Model checking and theorem proving. The ISC International Journal of Information Security 8(1):3–24

Rak M, Suri N, Luna J, Petcu D, Casola V, Villano U (2013) Security as a service using an sla-based approach via specs. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science, (CloudComp), pp 1–â6

Rios E, Mallouli W, Rak M, Casola V, Ortiz AM (2016) Sla-driven monitoring of multi-cloud application components using the musa framework. In: IEEE Inter-

national Conference on Distributed Computing Systems Workshops (ICDCSW), pp 55–60

Rios E, Rak M, Iturbe E, Mallouli W, et al. (2017) Sla-based continuous security assurance in multi-cloud devops

Sacramento L, Medeiros I, Bota J, Correia M (2018) Flowhacker: Detecting unknown network attacks in big traffic data using network flows. In: IEEE International Conference On Trust, Security And Privacy In Computing And Communications, pp 567–572

Sahay R, Meng W, Jensen CD (2019) The application of software defined networking on securing computer networks: A survey. Journal of Network and Computer Applications 131:89–108

Samuels CI, Syambas NR, Hendrawan, Edward IJM, Iskandar, Shalannanda W (2017) Service level measurement based on uptime data monitoring for rural internet access services in indonesia. In: International Conference on Telecommunication Systems Services and Applications (TSSA), pp 1–5

SAOX (2022)

Schwartz J, Kurniawati H (2019) Autonomous penetration testing using reinforcement learning. arXiv preprint arXiv:190505965

Sfondrini N, Motta G, You L (2015) Service level agreement (sla) in public cloud environments: A survey on the current enterprises adoption. In: International Conference on Information Science and Technology (ICIST), pp 181–185

Shacham H, Waters B (2008) Compact proofs of retrievability. In: Proc. of Asiacrypt, vol 5350, pp 90–â107

Shah U, Patel S, Jinwala D (2019) An ontological approach to specify conflicts among non-functional requirements. In: International Conference on Geoinformatics and Data Analysis, pp 145â–149

She R (2020) Survey on incentive strategies for mobile crowdsensing system. In: In-

ternational Conference on Software Engineering and Service Science (ICSESS), pp 511–514

Silva A, Silva K, Rocha A, Queiroz F (2019) Calculating the trust of providers through the construction weighted sec-sla. Future Generation Computer Systems 97:873–886

Standard S (2022) Security trust assurance risk

Stephen J, SSavvides, Seidel R, Eugster P (2014) Practical confidentiality preserving big data analysis. In: Proc. of USENIX HotCloud, pp 10–16

Su H, Pan J (2016) Crowdsourcing platform for collaboration management in vulnerability verification. In: Asia-Pacific Network Operations and Management Symposium (APNOMS), pp 1–4

Sun Y, Nanda S, Jaeger T (2015) Security-as-a-service for microservices-based cloud applications. In: IEEE International Conference on Cloud Computing Technology and Science, pp 50–57

Symantic (2019) Adapting the new reality of evolving cloud threats. https://resource.elq.symantec.com/e/f2

Tan CB, Hijazi MHA, Lim Y, Gani A (2018) A survey on proof of retrievability for cloud data integrity and availability: Cloud storage state-of-the-art, issues, solutions and future trends. Journal of Network and Computer Applications 110:75–86

Tang P, Qiu W, Huang Z, Lian H, Liu G (2020) Detection of sql injection based on artificial neural network. Knowledge-Based Systems 190:105528

Tedja A, Lim C, Ipung HP (2018) Detecting network anomalies in isp network using dns and netflow. In: International Conference on Innovation, Entrepreneurship and Technology, vol 2, pp 238–242

Tetali S, Lesani M, RMajumar, Millstein T (2013) Mrcrypt: Static analysis for secure cloud computations. In: Proc. of ACM SIGPLAN, pp 271–286

Tian H, Chen Y, Chang CC, Jiang H, Huang Y, Chen Y, Liu J (2015) Dynamic-hash-table based public auditing for secure cloud storage. IEEE Transactions on Services Computing 10(5):701–714

Tian H, Nan F, Chang CC, Huang Y, Lu J, Du Y (2019) Privacy-preserving public auditing for secure data storage in fog-to-cloud computing. Journal of Network and Computer Applications 127:59 – 69

Tosun A, De Donno M, Dragoni N, Fafoutis X (2021) Resip host detection: Identification of malicious residential ip proxy flows. In: IEEE International Conference on Consumer Electronics

Trapero R, Modic J, Stopar M, Taha A, Suri N (2017) A novel approach to manage cloud security sla incidents. Future Generation Computer Systems 72:193–205

Tu T, Rao L, Huan Z, Wen Q, Xiao J (2017) Privacy-preserving outsourced auditing scheme for dynamic data storage in cloud. Security and Communication Networks 2017:1–17

Verizon (2020) 2020-data-breach-investigations-report. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf

Viglianisi E, Ceccato M, Tonella P (2020) A federated society of bots for smart contract testing. Journal of Systems and Software 168:110647

Wang C, Wang Q, Ren K, Lou W (2010) Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings IEEE INFOCOM, pp 1–9

Wang C, Chow SSM, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. IEEE Transactions on Computers 62(2):362–375

Wang J, Peng F, Tian H, Chen W, Lu J (2019a) Public auditing of log integrity for cloud storage systems via blockchain. In: International Conference on Security and Privacy in New Computing Environments, Springer, pp 378–387

Wang Q, Wang C, Li J, Ren K, Lou W (2009) Enabling public verifiability and data

dynamics for storage security in cloud computing. In: Proceedings of the 14th European Conference on Research in Computer Security, pp 355â–370

Wang W, Shi X, Qin T (2019b) Encryption-free authentication and integrity protection in body area networks through physical unclonable functions. Smart Health 12(2):66–81

Wang W, Qin T, Wang Y (2020) Encryption-free data transmission and hand-over in two-tier body area networks. Elsevier Computer Methods and Programs in Biomedicine 192

Wang X, Tushar W, Yuen C, Zhang X (2020) Promoting users participation in mobile crowdsourcing: A distributed truthful incentive mechanism (dtim) approach. IEEE Transactions on Vehicular Technology 69(5):5570–5582

Wei L, Wu J, Long C (2020) A blockchain-based hybrid incentive model for crowdsensing. Electronics 9(2)

Wonjiga AT, Peisert S, Rilling L, Morin C (2019a) Blockchain as a trusted component in cloud sla verification. In: Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing Companion, pp 93–â100

Wonjiga AT, Rilling L, Morin C (2019b) Defining security monitoring slas in iaas clouds: the example of a network ids. Research Report RR-9263, Inria Rennes Bretagne Atlantique, pages 1–37

Wonjiga AT, Rilling L, Morin C (2019c) Security monitoring sla verification in clouds: the case of data integrity. Research Report RR-9267, Inria Rennes - Bretagne Atlantique, pages 1–29

Wool A (2010) Trends in firewall configuration errors: Measuring the holes in swiss cheese. IEEE Internet Computing 14(4):58–65

Xiao Y, Hao Q, Yao D (2019) Neural cryptanalysis: Metrics, methodology, and applications in cps ciphers. In: IEEE Conference on Dependable and Secure Computing (IDSC)

Yang X, Pei X, Wang M, Li T, Wang C (2020) Multi-replica and multi-cloud data public audit scheme based on blockchain. IEEE Access 8:144809–144822

Yang Y, Huang X, Liu X, Cheng H, Weng J, Luo X, Chang V (2019) A comprehensive survey on secure outsourced computation and its applications. IEEE Access 7:159426–159465

Zhao B, Tang S, Liu X, Zhang X (2021) Pace: Privacy-preserving and quality-aware incentive mechanism for mobile crowdsensing. IEEE Transactions on Mobile Computing 20(5):1924–1939

Zhao M, Geng Y (2019) Homomorphic encryption technology for cloud computing. Procedia Computer Science 154:73–83

Zhou L, Fu A, Yu S, Su M, Kuang B (2018) Data integrity verification of the outsourced big data in the cloud environment: A survey. Journal of Network and Computer Applications 122:1–15

Zhou Z, Zhang H, Yu X, Guo J (2015) Audit meets game theory: Verifying reliable execution of sla for compute-intensive program in cloud. In: IEEE International Conference on Communications (ICC), pp 7456–7461