PROTECTING AUDIT DATA USING SEGMENTATION-BASED ANONYMIZATION FOR MULTI-TENANT CLOUD AUDITING (SEGGUARD)

Momen Oqaily

A thesis in The
Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science
in Information Systems Security at
Concordia University
Montréal, Québec, Canada

August 2018

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Momen Oqaily**
Entitled:	**Protecting Audit Data Using Segmentation-based Anonymization for Multi-tenant Cloud Auditing (Seg-Guard)**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Abdessamad Ben Hamza	______________________	Chair
Dr. Emad Shihab	______________________	Examiner
Dr. Mohammad Mannan	______________________	External Examiner
Dr. Lingyu Wang	______________________	Supervisor
Dr. Mourad Debbabi	______________________	Co-supervisor

Approved ______________________________________

Chair of Department or Graduate Program Director

__________ 2018 ______________________________

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

Abstract

Protecting Audit Data Using Segmentation-based Anonymization for Multi-tenant Cloud Auditing (SegGuard)

Momen Oqaily

With the rise of security concerns regarding cloud computing, the importance of security auditing, conducted either in-house or by a third party, has become evident more than ever. However, the input data required for auditing a multi-tenant cloud environment typically contains sensitive information, such as the topology of the underlying cloud infrastructure. Additionally, audit results intended for one tenant may unexpectedly reveal private information, such as unpatched security flaws, about other tenants. How to anonymize audit data and results in order to prevent such information leakage is a novel challenge that has received little attention. Directly applying most existing anonymization techniques to such a context would either lead to insufficient protection or render the data unsuitable for auditing. In this thesis, we propose *SegGuard*, a novel anonymization approach that protects the sensitive information in both the audit data and auditing results, while assuring the data utility for effective auditing. Specifically, *SegGuard* prevents cross-tenant information leakage through per-tenant encryption, and it prevents information leakage to auditors through an innovative way of applying property-preserving anonymization. We apply *SegGuard* on audit data collected from an OpenStack cloud, and evaluate its effectiveness and efficiency using both synthetic and real data. Our experimental results demonstrate that *SegGuard* can reduce information leakage to a negligible level (e.g., less than

1% for an adversary with 50% pre-knowledge) with a practical response time (e.g., 62 seconds to anonymize a cloud virtual infrastructure with 25,000 virtual machines).

Acknowledgments

I would like first to express my special thanks of gratitude to my thesis advisor Prof. Lingyu Wang, as well as to my co-supervisor Prof. Mourad Debbabi. They gave me the golden opportunity and helped me in doing a lot of research and I came to know about so many new things. Their continuous guidance, support and encouragement helped me the most to finish this thesis work. They always understand my needs and problems, provides me with endless help and try there best to give me generous and wise pieces of advice. Furthermore, Prof. Lingyu Wang was my godfather during this long journey. His way of thinking, inspiring words and kindness changed my life a lot not only at the research level but at the personal level also. Personally speaking, I feel very proud and unique to be one of his students.

I would also like to thank my lab mates and researchers from Ericson represented by the Audit Cloud Ready members. Special thanks to Amir Alimohammadifar, Suryadipta Majumdar, Meisam Mohamady and Yosr Jarraya from Ericson, who were very cooperative with me from the very beginning of my master's program and gave me a lot of their time and effort. In all honesty, their invaluable support and guidance helped me a lot for completing my master's thesis.

I am also deeply indebted to my respected teachers and all other faculty members of the CIISE department. My words have no bounds in expressing their professionalism and kindness that I felt during the courses that I attended during my master's.

Finally, I submit my heartiest gratitude to my parents and to my family members for their love, support and continuous encouragement from the beginning of my years of study, they all kept me going. I wish that by accomplishing this achievement I bring some of the thoughts in your minds to a reality and I promise you to do my best to achieve more and more. I also humbly extend my thanks to all concerned persons who supported me in this regard.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing environments are becoming widely adapted, which leads to an increase in the fear factor about its security and privacy concerns among its tenants. To ensure the security of a multi-tenant cloud, a cloud service provider (CSP) usually performs security auditing of their virtual infrastructure, either in-house by a cloud security admin or by an appointed third-party auditor, in order to detect threats and vulnerabilities as well as to assess the compliance of the cloud environment with respect to security standards and regulations [24]. When done properly, security auditing may help the CSP in remedying security breaches and compliance violations [48].

On the other hand, auditing may pose unique security challenges in a multi-tenant cloud environment. First, most auditing tasks would require the CSP to grant auditors accesses to data about both the cloud physical infrastructure and the tenants' virtual infrastructures. Second, auditing results intended for one tenant may unwillingly include data about other tenants due to the multi-tenancy nature of clouds. In both cases, the data may carry sensitive and private information, such as private IP addresses, virtual network topologies, and inter-tenant communication patterns. Furthermore, such data may contain explicit tenant identifiers, which are used to relate virtual resources to specific cloud customers owning those resources. Those identifiers, used in conjunction with auxiliary knowledge (e.g., customer databases), may be used to connect the sensitive information contained in audit data or audit results to real world customers, leading to violation of privacy regulations (such as the newly adopted GDPR [11]). Additionally, there exist more sensitive attributes, e.g., routing rules, security group rules, which may be captured by an adversary to understand the security policies used by a given tenant and abuse it to parasitize or even disrupt services of other tenants or to take down some services supplied by the cloud service provider [13]. On the other hand, auditing results can only be reliable if the data provided to the auditor is accurate and complete. Thus, preserving utility with respect to the security auditing tasks is of a paramount importance in the context of cloud auditing.

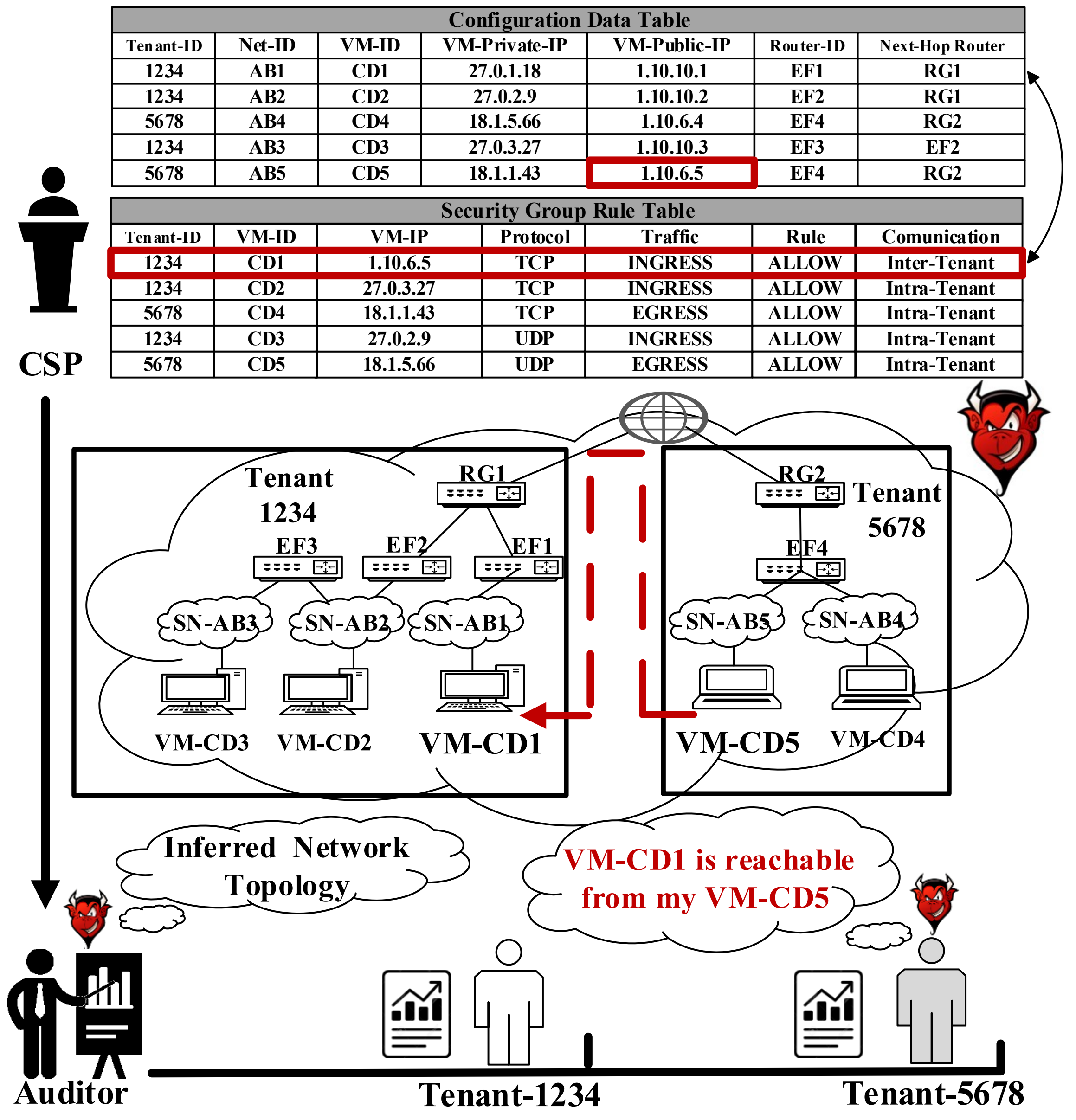
To the best of our knowledge, the issue of preserving privacy for security auditing in clouds has largely been ignored so far in the literature. There exist cloud security auditng approaches, which typically ignore the privacy issues altogether and consequently assume a trusted central auditor who does not disclose any sensitive information about other entities and the audit outputs cannot be misused. However, this assumption is impractical and somewhat threatening especially in the cloud environment since both virtual network infrastructure and physical resources could be shared among different tenants with different levels of trust. On the other hand, most existing privacy preserving techniques in the context of general purpose network traces and data anonymization, are applying stringent privacy protection mechanisms. For instance, they can not anonymize network topology, but only low level information, e.g., IPs, and they cannot preserve important relationships between data attributes. However, scarifying sensitive information from the anonymized data or intentionally breaking the inherent relationships, are defeating the auditing purposes. Hence, any loss in the data utility leads to inconsistency and incorrectness in the auditing results. In the following, we further motivate these through an example.

In the following section, we demonstrate such threats to security and privacy from audit data and audit results, as well as the importance of utility preservation for auditing through an example.

1.1 Motivation

Figure 1 illustrates two tables containing sample audit data collected from the configuration databases of different services of OpenStack [37] in a multi-tenant cloud environment. For simplicity, we only show data belonging to Tenant1 (*Tenant-ID: 1234*) and Tenant2 (*Tenant-ID: 5678*). As an example auditing task, we consider the verification of reachability between all pairs of virtual machines (VMs). The upper table of Figure 1, called the *configuration data table*, contains attributes about the virtual networks and their VMs, such as the tenant identifier, the virtual network identifier, the VM identifier, the private and public IPs corresponding to that VM, the virtual router ID connecting the network to other subnets, and the next hop router. The lower table of Figure 1, called the *security group rule table (SGR)*, shows the security group rules associated with the VMs shown in the above configuration table. Sharing such configuration data as-is may give rise to following security and privacy concerns.

1. Upon receiving the configuration table, an auditor can aggregate the data to construct the entire topology of the cloud infrastructure as shown in Figure 1. Any leakage of such information by a dishonest third party auditor or careless insider might cause serious consequences to the CSP, including further attacks enabled by knowledge about the topology (e.g., topology poisoning and DoS)



Configuration Data Table						
Tenant-ID	Net-ID	VM-ID	VM-Private-IP	VM-Public-IP	Router-ID	Next-Hop Router
1234	AB1	CD1	27.0.1.18	1.10.10.1	EF1	RG1
1234	AB2	CD2	27.0.2.9	1.10.10.2	EF2	RG1
5678	AB4	CD4	18.1.5.66	1.10.6.4	EF4	RG2
1234	AB3	CD3	27.0.3.27	1.10.10.3	EF3	EF2
5678	AB5	CD5	18.1.1.43	1.10.6.5	EF4	RG2

Security Group Rule Table						
Tenant-ID	VM-ID	VM-IP	Protocol	Traffic	Rule	Comunication
1234	CD1	1.10.6.5	TCP	INGRESS	ALLOW	Inter-Tenant
1234	CD2	27.0.3.27	TCP	INGRESS	ALLOW	Intra-Tenant
5678	CD4	18.1.1.43	TCP	EGRESS	ALLOW	Intra-Tenant
1234	CD3	27.0.2.9	UDP	INGRESS	ALLOW	Intra-Tenant
5678	CD5	18.1.5.66	UDP	EGRESS	ALLOW	Intra-Tenant

Figure 1: An example of configuration data collected from different OpenStack cloud services about VMs and their networks of two different tenants (e.g., ID: 1234 and ID: 5678). The upper (resp. lower) table represents the configuration of the tenants' topologies (resp. VMs' security group rules).

[31].

2. The auditing results from reachability analysis intended for one tenant may also contain sensitive information about other tenants. For example, the auditing results may reveal that a Tenant1 VM *CD1* is reachable from a Tenant2 VM *CD5* (as shown by the dashed line in Figure 1). Specifically, VM *CD1* (first record of configuration table) is associated with the security group rule (first record of SGR table) that allows the remote connection from *1.10.6.5* (fifth row in configuration table), which is the public IP of VM *CD5.* Disclosing such information may potentially breach the service level agreement (SLA) of Tenant 1 and invite further attacks from a (malicious) user of Tenant2 by exploiting the discovered reachability between *CD1* and *CD5.*

3. At the same time, sufficient utility must be preserved in the data to allow an auditor to perform the desired reachability analysis. For instance, the configuration table shows that *CD2* and *CD3* belong to the same tenant (i.e., Tenant-ID: 1234), and the second record of the SGR table shows that *CD2* is reachable by *CD3* because its security group rule allows traffic from IP *27.0.3.27.* Such a reachability analysis will be possible if the common Tenant-ID attribute can be used to join the two tables together. For this reason, randomization or encryption techniques that destroy such an equality property will not be applicable. On the other hand, we notice that, as long as certain properties (e.g., equality between Tenant-IDs and shared prefixes between IP addresses) are preserved, we would be able to perform such auditing tasks on anonymized (encrypted) data.

To that end, many works have focused on property-preserving anonymization of network trace data (a more detailed review of related works will be given in section 2.2). For example, the so-called prefix-preserving anonymization technique anonymizes IP addresses by replacing them with random-looking IPs, while preserving the shared prefixes such that IPs within the same subnet would remain so after the anonymization [21]. However, such techniques are known to be vulnerable to the so-called *semantic* attacks in which adversaries may expand their prior knowledge about certain IPs to other IPs sharing the same prefixes even if all the IPs have been anonymized. Other solutions tradeoff utility for security (e.g., suppression and bucketization [40], and aggregation with perturbation [34]). Directly applying those existing techniques to address the unique challenges of anonymizing audit data and audit results, as demonstrated above, would either suffer from the same semantic attacks or fail to meet the utility requirements of auditing (e.g., suppression, bucketization, or aggregation and perturbation may all prevent the reachability analysis in our example).

1.2 Thesis Statement

In this thesis, we present, *SegGuard*, a novel anonymization approach that prevents the sensitive and private information in both audit data and audit results from being disclosed to semi-trusted auditors and tenants, while preserving sufficient data utility to facilitate effective security auditing. Our key ideas are twofold. First, we employ per-tenant encryption to ensure each tenant can benefit from useful auditing results (e.g., a VM is reachable from other tenants' VMs) without learning details about other tenants' resources (e.g., the exact IDs of other tenants or their VMs). Second, we propose an innovative segmentation-based, layered encryption technique to ensure auditors can perform the desired auditing tasks without learning either the original configuration data or the auditing results. Specifically, we first divide the original audit data into multiple segments, and then let the CSP and the auditor each apply a certain number of iterations of property-preserving encryption (PPE) [35] on each segment. Our design ensures that, the total number of iterations of PPE (applied by both the CSP and auditor) will be different for most segments (which means the property is not preserved between those segments, namely, “fake” segments) except a few selected segments (which receive the same number of iterations of PPE and hence have their property preserved, namely, the “real” segments). Finally, the auditor will perform the auditing tasks over all the segments, while s/he cannot tell which of those segments (and hence their results) are real since the number of iterations of PPE applied by the CSP is kept secret from him/her. On the other hand, the CSP can secretly extract and integrate all the real auditing results and send them to the tenants.

1.3 Contributions

Our main contributions in this work are as follows.

- To the best of our knowledge, this is the first effort addressing the unique challenges of protecting the security and privacy of both audit data and audit results in a multi-tenancy cloud environment.
- Our per-tenant encryption solution allows tenants to benefit from auditing results without breaching other tenants' privacy, which may help ease the privacy concerns of both CSP and tenants in adopting security auditing solutions.
- Our segmentation-based, layered encryption technique allows semi-trusted auditors to perform auditing on anonymized data without learning either the detailed configuration information or auditing results, which reduces the risk of leaking such sensitive information and also makes it easier for CSP to outsource the auditing tasks to specialized third party providers.

- We present detailed methodology, describe our implementation based on OpenStack, and evaluate our solution through experiments using both synthetic and real data. The results demonstrate our solution can reduce the level of information leakage to a negligible level with practical response time (e.g., less than 1% information leakage for adversaries armed with 50% pre-knowledge, and less than 62 seconds required for anonymizing a cloud infrastructure with 25k VMs).

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background information and reviews the related works. In Chapter 3 we provide our threat model. Chapter 4 presents the proposed solution. Chapter 5 discusses the implementation details. Chapter 6 analyzes the security, privacy, and utility, and discusses other aspects of our approach. Chapter 8 gives our experimental results. Finally, Chapter 9 discusses the limitations and concludes this work.

Chapter 2

Preliminaries and Related Work

The first part of this chapter provides preliminaries of our work and the second part discusses literature related to our work.

2.1 Preliminaries

This section provides an extensive overview and give background information about some exisiting anonymization techniques and auditing method that we use in our work.

2.1.1 Property Preserving Encryption (PPE)

In this work, we focus on auditing tasks that can be performed on anonymized data. For example, as demonstrated in Chapter 1, the reachability analysis relies on the equality (between Tenant-IDs, VM-IDs, etc.) and prefix preserving properties (between the IP addresses). The equality property can be achieved through deterministic encryption (i.e., the same plaintext always leads to the same ciphertext under the same key), whereas the prefix preserving property can be achieved through special encryption techniques (detailed below). Other auditing tasks may require additional properties and PPE techniques [35, 2], such as order-preserving encryption (i.e., ciphertext will preserve the order of the plaintext [5]). In the remainder of this work, we will focus on the equality and prefix-preserving properties but our solution can be adapted to incorporate other PPE techniques.

To facilitate further discussions, we briefly discuss one of the cryptography-based PPE techniques, influenced by [51]. An anonymization function F is said to be *prefix-reserving*, if, for any two IP addresses x and y originally sharing a K-bit prefix, their anonymized versions, namely, $F(x)$ and $F(y)$, also share a K-bit prefix. The prefix preserving function F is also deterministic (i.e., the same address appearing in different traces will be mapped to the same anonymized address under the same key, which allows consistency in the anonymization process). Also, F must satisfy the canonical form [51]: given that $a = a_1, a_2 \ldots a_{32}$ and $F(a) = a_1', a_2' \ldots a_{32}'$, we have $a_i' = a_i \oplus f_{i-1}(a_1, a_2, \ldots, a_{i-1})$ for $i \in \{1, 2, \cdots, 32\}$, where f_i is a cryptographic function that takes as input a bit string of length $(i-1)$ and returns a single bit. That is, the i^{th} bit is anonymized based on a key and the preceding $(i-1)$ bits in order to satisfy the prefix-preserving property. This scheme is known to be immune against chosen-plaintext attacks if the encryption function is either stateful or randomized [27].

2.1.2 Compositional Auditing

As mentioned in Chapter 1, we divide the audit data into segments on which the auditing task will be performed, which makes our work more suitable for auditing tasks that can be performed in a divide-and-conquer manner [18]. Specifically, instead of auditing the entire input data in one shot, the data can be divided into smaller chunks to be audited separately, before the partial auditing results are combined to yield the same result as if the data was audited in one shot. For example, auditing the reachability between communicating devices (e.g., VMs) may be performed for each pair in the dataset separately, then the results can be aggregated. Many other common security properties are also compositional, such as virtual infrastructure-level security properties (e.g., VM co-residence, cross-layer port consistency and common ownership [26, 8, 36]), network-level security properties (e.g., loops, black-holes and incorrect snapshot [28, 54]), and application-level properties (e.g., functional and security-related [25]). Finally, since our divide-and-conquer approach resembles that found in most existing parallel processing platforms (e.g., MapReduce [19]), a natural extension is to integrate our approach with such platforms to further improve its efficiency and scalability.

2.2 Related Work

The most widely used techniques for anonymizing network data are truncation, randomization, quantization and pseudonymization. Truncation and randomization [9] effectively destroy the semantics of the field they are applied to. One example is the payload of packets, which might contain usernames and passwords that are removed from the data as a standard practice. Quantization techniques [15], such as limiting the precision of timestamps, are applied to reduce the information gained about the identity of the workstations from timing attacks [38]. The most widely used technique, pseudonymization [6], replaces IPs found in the data with linkable, prefix-preserving pseudonyms. These pseudonyms preserve the hierarchical relationships found in the prefixes of the original IPs. The underlying goal is to enable the analysis of packets generated by hosts, or whole prefixes, without providing the actual IPs. However, *SegGuard* does not replace or eliminate any data component that may affect the auditing process. Furthermore, *SegGuard* preserves the utility and the relationship in the real segments, such that their auditing results are error free.

The problem of data outsourcing and privacy leakage has been addressed extensively. Naveed et al. [35] studied the security of databases encrypted using PPE and presented different types of attacks that allow an adversary to decrypt a large portion of the encrypted data. The attack discussed therein was proposed by [7], where it has been shown that an adversary with a pre-knowledge can deanonymize selected addresses or subnets. However, we implemented and applied the same attack to test our scheme and we found out, as shown in Chapter 8, that our solution is immune against this type of attacks. Many information disclosure and information loss limitation methods have been discussed in [20]. Samarati and Sweeney [42, 41] define a complete framework for information disclosure control through the definition of K-anonymity approach. Gehrke [33] proposes L-diversity approach, providing a certain level of privacy even when the publisher does not know what kind of knowledge the adversary possesses. However, K-anonymity and L-diversity do not prevent attribute disclosure, especially when the table has multiple records belonging to one individual [33, 46, 50]. The (k,j)-obfuscation technique was introduced in [39]. Therein, authors addressed the issue of sensitive data obfuscation in network flows by introducing protection guarantees under realistic assumptions about the adversary's knowledge. In (k,j)-obfuscation, the data utility and information accuracy remain challenging, as the shared data has been heavily sanitized (i.e., from each k flows, having similar fingerprints, one flow is blurred). Chen et al. [12] address the problem of privacy-preserving quantification of real network reachability across different domains by preserving the privacy of access control configuration and access control lists only (layer three devices). Ciriani et al. [49] tried to provide privacy guarantees when sensitive information is stored, processed or shared to a second party through data fragmentation and encryption. Their approach makes data ambiguous and unintelligible using encryption. Contrary to all the above approaches, *SegGuard* preserves the utility of the data to be audited and returns valid results.

The loss of information/accuracy in data privacy preserving approaches due to the trade-off between privacy and utility has always been considered as a major issue. However, this loss cannot be tolerated in the case of auditing since any loss in data utility will affect the auditing results accuracy. This problem has been tackled in [12] and [14]. However, the main problem in the solution presented in [12] is that it only considers the access control lists and routers configuration not covering the virtual switches and virtual machines in the cloud environment. Hence, they do not verify the topology of the verified network. In [14] they are using data encryption to make the data unintelligible, and fragmentation as a way to break sensitive associations between information. However, there proposed solution used in data storing applications where the stored data is completely unavailable without accessing the encryption key. Finally, there are several works (e.g., [32, 3, 4]) that have been proposed to verify the virtual infrastructure logs in the cloud. Most of them considering the verification without focusing on the privacy concerns resulting from such process, or they keep the door open for future work. For instance, in [48] authors propose a scalable system for verifying cloud-wide VM-level isolation. This system returns, as a result, the pairs of VMs that are reachable even if they belong to different tenants. While this system assumes that the verification results do not disclose tenants' sensitive information, *SegGuard* builds on the assumption that audit results may disclose any sensitive information across parties involved in the auditing and preserves it by performing per-tenant data anonymization. In Table 1, we summarize the main weaknesses of the existing solutions.

To the best of our knowledge, no previous work proposed a cross-tenant privacy preserving approach that not only preserves the privacy of the data but also of the topology and relationships (e.g., topology of the virtual infrastructure) while ensuring its utility.

Approach	Limitations		
	Semantic Attacks	Privacy Leaking	sacrificing Utility
Prefix-preserving	✓		
Truncation and Randomization		✓	✓
Quantization and Pseudonymization	✓		✓
Shifting and Permutation			✓
Hashing and Encryption		✓	✓

Table 1: Summary of existing anonymization solutions and their main weaknesses.

Chapter 3

Models

In this chapter, we define our threat model and identify the trust relationships between different stakeholders to be achieved with our proposal.

3.1 System Model and Trust Relationships

SegGuard threat model is similar to those used in existing works [45, 43, 55, 1]. First, we define the stakeholders involved in the cloud service model as follows:

1. The cloud service provider (CSP): This is the entity providing paid services and has a significant interest in protecting their reputation and building trust to attract customers. We also differentiate between the CSP and the cloud administrators as detailed below in the trust relationships.

2. The cloud tenants: Those are customers of the CSP and are concerned with the security of their deployed virtual resources and the privacy of their sensitive data.

3. The cloud auditor: This can be either the cloud administrators or third-party auditors who have the expertise and capabilities needed to perform auditing.

Our assumptions about the trust relationships between those stakeholders are detailed in the following and depicted in Figure 2.

1. Tenants usually have to trust the CSP for protecting the security and privacy related to both the audit data and the auditing results as part of the fulfillment of the service level agreement (SLA).

2. A tenant (and CSP) does not trust other tenants with the received auditing results (e.g., a tenant might misuse the results to launch attacks against other tenants).

3. Tenants do not trust the auditors (cloud administrators or third-party), particularly, to have access to their sensitive data and confidential information (e.g., private IPs, virtual infrastructure topologies, etc.).

4. CSP does not trust the auditor to have access to non-protected audit data and audit results as in [47, 10]. More precisely, auditors might have the technical means and the financial motivation to misuse their privileges in order to extract information from the audit data or from the audit results if they are allowed to do so.

3.2 In-scope Threats and Adversarial Model

SegGuard aims at protecting information related to security and privacy that might be extracted from the original audit data and the auditing results by an honest but curious entity (auditor or tenant). More precisely, similar to existing works [23, 10, 47], we assume an adversary who will follow given protocols to access anonymized data and perform audit tasks, while s/he is curious enough to infer sensitive information

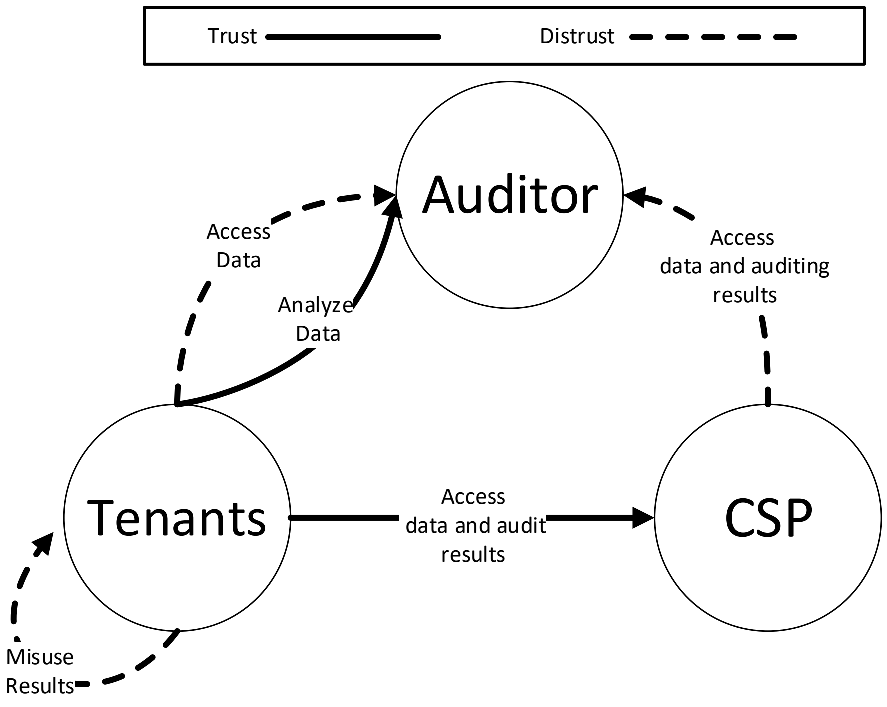


Figure 2: A summary of trust relationships.

about the cloud infrastructures or the tenants, if the protocols provide him/her such an opportunity. We can distinguish the following types of attacks:

1. *Individual attacks:* This attack can be done by either a tenant or an auditor. The goal of such attacks is to recover individual information about a specific tenant, e.g., an attacker can identify a particular tenant based on the cloud tenant identifier, and infer his/her private information, such as private IPs and tenants' relationships. Such an attack could either cause privacy breaches or enable further more severe attacks.

2. *Aggregate attacks:* This attack can be done by an auditor. The goal of such attacks is to recover general information about a subset of (or the entire) cloud infrastructure, e.g., tenants' virtual network topologies, known vulnerabilities and communication relationships between different tenants. Such an attack

could also have both security and privacy consequences.

3.2.1 Adversarial Knowledge

We consider adversaries with prior knowledge about the system and its data. Examples of information that may potentially be collected by the adversary include: **i)** publicly available information (e.g., general information about the cloud and its customers, system versions, current and planned usage); **ii)** prior resources (e.g., IPs that have been leaked through previous breaches or attacks); **iii)** any information collected from other resources or from the system itself (e.g., details about applications and services to which the adversary has access to). Since a tenant or auditor may easily possess all such information about a victim, it is mandatory to consider such a practical adversarial model. Our methodology and experiments both take this into consideration, and we study different cases of adversarial knowledge.

3.2.2 Out-of-Scope Threats

We do not consider malicious adversaries who deviate from given protocols. We also consider the integrity or availability issues related to audit reports out-of-scope. We focus on information leaked from audit data and audit report, and do not consider attacks from other sources, e.g., side-channel attacks. Finally, we do not defend against distrusted CSP.

Chapter 4

SegGuard System

This chpater details *SegGuard* methodology. We first present the main idea of our approach then we provide an overview of its steps and finally we further detail each step.

4.1 Main Idea

Before we delve into the details of *SegGuard*, we first build intuitions about its main ideas by considering only three IP addresses from Figure 1. These are shown in Table 2 (second column under “Original IP”). As discussed in Chapter 1, the reachability analysis can be performed on those IPs as long as their shared prefixes are preserved (we will ignore other attributes for now). Roughly speaking (many details are omitted and will be provided later in this chapter), *SegGuard* works as follows.

	Original IP	**Encrypted with K_T**	V_0	**Encrypted with K_{An}**
Tenant1	1.10.10.1	93.14.36.9	1	45.17.7.9
Tenant1	1.10.10.2	93.14.15.14	2	132.6.4.66
Tenant1	1.10.6.5	93.14.22.8	3	201.47.96.23

Table 2: The CSP side.

	Received	V_1	**Copy1**	V_2	**Copy2**	V_3	**Copy3**
Tenant1	45.17.7.9	2	149.118.33.23	3	57.188.165.42	1	30.11.21.17
Tenant1	132.6.4.66	1	149.118.74.35	0	132.6.4.66	2	51.18.50.55
Tenant1	201.47.96.23	3	85.50.44.118	1	57.188.34.22	1	51.18.20.25

Table 3: The auditor side (where shaded IPs are utility-preserving and lead to valid auditing results).

1. First, the CSP encrypts the original IP addresses using a key in which s/he agrred on it with the each tenant (K_T) and a prefix-preserving encryption algorithm, as shown in the third column of Table 2. Since all the later steps will be layered upon this initial per-tenant encryption, and key K_T is never shared with other tenants, this step will ensure no cross-tenant information leakage from the auditing results.

2. Next, the CSP generates a random integer vector V_0 (fourth column), and the second key K_{An}. It then encrypts each IP in third column (already encrypted under K_T) iteratively, where each element of vector V_0 indicates the number of iterations applied, e.g., the first IP is encrypted once, and the second twice, etc. We can notice that, since the number of iterations is different for each IP, the results (shown in the last column) no longer contain any shared prefixes. Those IPs in the last column are then sent to the auditor together with three new vectors, V_1, V_2 and V_3 (generated similarly as V_0, and shown in Table 3).

3. Table 3 shows what happens at the auditor side. The auditor also applies the same prefix-preserving encryption for different iterations based on the V_1, V_2, and V_3 vectors, similar to how the CSP has used V_0. The resultant IPs are shown in the columns next to their corresponding vectors. As those iterations add up, shared prefixes start to appear, as shown with the gray shade, e.g., in the fourth column, the first two IPs share the same prefix 149.118, since

they have both been encrypted three times now (i.e., $1 + 2 = 3$ and $2 + 1 = 3$ for the first and second IPs, respectively). We can observe that, those vectors are chosen in such a way, that exactly one pair of IPs share prefixes in every copy, namely, *utility-preserving* IPs. However, although not shown in this toy example, real audit data would also include a larger number of other IPs which also share prefixes here at the auditor side, while they actually do not share prefixes in the original IP column.

4. Next, the auditor performs the auditing task on each copy and obtains the results for all the IPs. From above discussions, we know that, since there is only one pair of utility-preserving IPs (the shaded ones) in each copy, the results are only valid between those IPs. However, the auditor will not have this knowledge, since the auditor does not know V_0, and there exist many other pairs IPs that also share prefixes in each copy as mentioned above.

5. Back to the CSP side, the CSP knows which IPs are utility-preserving since s/he knows V_0, so s/he extracts only the valid results from each copy, integrates those results, and sends them to the tenants.

6. Each tenant decrypts his/her own IPs using K_T to reveal the auditing results, with other tenants' IPs still encrypted.

4.2 Overview

To realize the above ideas, the *SegGuard* approach encompasses ten steps as depicated in Figure 3.

Steps 1-2: Initially, each tenant shares his/her key with the CSP (Step 1), who encrypts tenants' data using their supplied keys (Step 2), as will be detailed in Section 4.3.1.

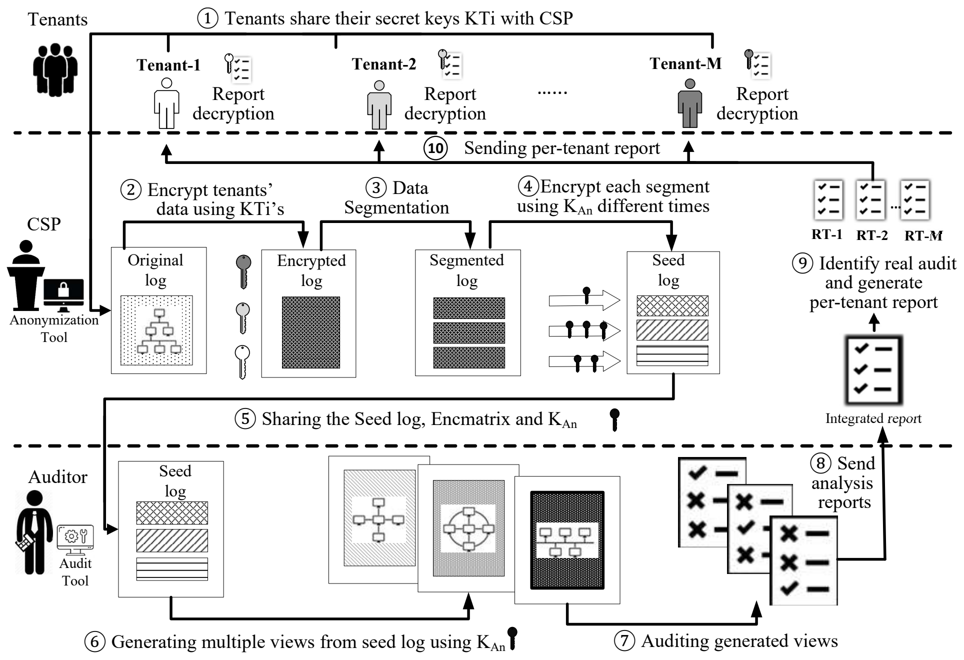

Tenants
① Tenants share their secret keys KTi with CSP
Tenant-1
Report decryption
Tenant-2
Report decryption
Tenant-M
Report decryption
⑩ Sending per-tenant report
CSP
② Encrypt tenants' data using KTi's
③ Data Segmentation
④ Encrypt each segment using K_An different times
RT-1
RT-2
RT-M
⑨ Identify real audit and generate per-tenant report
Original log
Encrypted log
Segmented log
Seed log
Anonymization Tool
Integrated report
⑤ Sharing the Seed log, Encmatrix and K_An
Auditor
Seed log
Audit Tool
⑧ Send analysis reports
⑥ Generating multiple views from seed log using K_An
⑦ Auditing generated views

Figure 3: An overview of *SegGuard* approach.

Steps 3-4: Next, the CSP aggregates all tenants' data, divides it into segments (Step 3) and encrypts each segment (using key K_{An} shared between CSP and the auditor) for different iteration based on a randomly generated vector (i.e., V_0 in the previous example). The output of this step is called the seed log (Step 4), as will be detailed in Section 4.3.2 and Section 4.3.3.

Steps 5-6: The CSP shares the generated vectors and seed log with the auditor (Step 5). The auditor then applies those vectors to the seed log in order to generate multiple views (called copies in Table 3) in which "real" (i.e., utility preserving) segments are hidden among "fake" segments (Step 6). The fake segments in those views help to hide from the auditor not only the sensitive attributes (e.g., the IPs) but also the virtual topology, as will be detailed later. On the other hand, the real segments scattered among those views help to guarantee the data utility, as the auditor will unknowingly perform the requested auditing tasks on all the real segments, whose union is exactly a utility-preserving copy of the originally anonymized data, as will be detailed in Section 4.3.4.

Steps 7-9: The auditor then performs auditing on the generated views (Step 7) and sends the results to the CSP (Step 8). Then, the CSP identifies the real audit results from each report, integrates them, and generates a per-tenant report (Step 9), as will be detailed in Section 4.3.5.

Step 10: Finally, once each tenant receives the audit report, s/he decrypts his/her own data inside the auditing report (Step 10), as will be detailed in Section 4.3.5.

4.3 Architecture

This Chapter describes the steps of *SegGuard* mechanism in details.

4.3.1 Per-Tenant Encryption

Steps 1-2 in Figure 3 provide the first layer of anonymization to preserve the privacy of the sensitive attributes of each tenant. They are responsible for i) agreeing with tenants on a secret key K_{Ti} and an initialization vector IV_i from each tenant via a trusted channel (Step 1 in Figure 3), and ii) encrypting each attribute of audit data related to tenant T_i using (K_{Ti}, IV_i) with the appropriate algorithm (Step 2 in Figure 3) depending on the type of data (e.g., prefix-preserving encryption for IPs or deterministic encryption for identifiers). The aggregated and encrypted data is then passed to the next step.

Example 1. Table 4 illustrates the result of encrypted data of the configuration table in Figure 1 using K_{T1} and K_{T2}, respectively. IP addresses are encrypted while preserving their prefixes.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
9998	X1X	VVV	66.22.10.3	18.12.12.14	WW1	YSE
9998	X3C	MX2	66.22.7.66	18.12.12.16	WW3	YSE
5554	ZQA	SQ1	98.6.36.82	101.2.42.9	WW4	JHQ
9998	X1X	GTQ	66.22.17.34	18.12.12.7	WW1	EQW
5554	CCY	TT2	98.6.41.13	101.2.42.13	WW5	JHQ

Table 4: Encrypted data corresponding to the original configuration table of Figure 1 performed by the CSP using K_{T1} (unshaded rows) and K_{T2} (shaded rows).

4.3.2 Data Segmentation

Step 3 provides the second layer of anonymization to protect the relationships between data attributes, and hence the virtual topology. It takes two inputs: i) the aggregated encrypted audit data of all tenants from the per-tenant encryption; ii) the parameters, i.e., the total number of segments (N_{seg}) and the number of real segments per view ($N_{seg-view}$), related to segmentation as chosen by the CSP, details about selection of these parameters are discussed in Section 7.1. The CSP then performs the following operations:

1. **Computing number of views:** The CSP computes the number of views N_{views} based on the following formula:

$$N_{views} = \frac{N_{seg}!}{N_{seg-view}! \times (N_{seg} - N_{seg-view})!} \tag{1}$$

   For a given value of $N_{seg-view}$, each view generated at the auditor would contain N_{seg} segments in total where $N_{seg-view}$ are real segments and the rests (i.e., $N_{seg} - N_{seg-view}$) are fake. For example, if we consider the case where $N_{seg-view} = 2$ and $N_{seg} = 5$ (as in Table 4), then, based on Equation 1, the auditor has to generate 10 views.

2. **Data segmentation:** First, the data is sorted based on the tenant ID and

network ID. Then, it is parceled among the N_{seg} segments in a round robin fashion [29], (Step 3 in Figure 3) to ensure that each network spreads out to minimize data relationships (e.g., network topology) leakage in the views generated later by the auditor. In *SegGuard*, the (anonymized) tenant ID is used across different steps, i.e., data segmentation, auditing and result integration for the purpose of identifying tenant's assets. The output of this step is the segmented log data, denoted by *SegLog*.

Example 2. Table 5 shows the result of sorting and segmenting the data, *SegLog*, where a record in Table 4 represents one segment.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
9998	X1X	VVV	66.22.10.3	18.12.12.14	WW1	YSE
5554	ZQA	SQ1	98.6.36.82	101.2.42.9	WW4	JHQ
9998	X3C	MX2	66.22.7.66	18.12.12.16	WW3	YSE
5554	CCY	TT2	98.6.41.13	101.2.42.13	WW5	JHQ
9998	X1X	GTQ	66.22.17.34	18.12.12.7	WW1	EQW

Table 5: *SegLog* corresponding to the encrypted and segmented data of Table 4 performed at the CSP side.

4.3.3 Utility Preserving Anonymization

The CSP first generates a set of utility parameters, (i.e., V_0 - V_3 in the example discussed in Chapter 5), to be used to ensure the utility of *SegLog* and the final audit results while protecting the data. Then based on these parameters, the CSP applies a third (and final) set of transformations on *SegLog* before transferring it to the auditor (Step 4 in Figure 3). This is detailed in the following:

Utility Preserving Parameters. The generated utility preserving parameters can be classified into two categories:

-Secret parameters: i) a random vector V_{Random} of dimension $(N_{seg} * 1)$, where elements are unique, randomly generated integer values and ii) set of vectors $\{VP_i\}_{i \in N_{views}}$ each of size $(N_{seg} * 1)$ where elements are integer values such that in each vector only two elements are equal and at least one of those equal indices is different for any pair of vectors. Algorithm 1 is used to generate vectors $\{VP_i\}_{i \in N_{views}}$.

-Parameters shared with the auditor: i) a matrix $EncMatrix$ of size $(N_{seg} \times N_{views})$ and ii) an encryption key K_{An}. Elements of the $EncMatrix$ are computed using V_{Random}, $\{VP_i\}_{i \in N_{views}}$ based on the following equation:

$$EncMatrix[i][j] = VP[i][j] - V_{Random}[j] \tag{2}$$

Algorithm 1 GenerateVP

```
Input: $N_{views}$, $N_{seg}$
Output: $VP[N_{views}][N_{seg}]$
$PtrX \leftarrow 2, PtrY \leftarrow 1, intVP[][] \leftarrow \emptyset, random[] \leftarrow \emptyset$
for $(int\ i = 1; i <= N_{views}; i++)$ do
    for $(int\ j = 1; j <= N_{seg}; j++)$ do
        $random[j] = \textbf{Rand}()$                    ▷ Generate Random Number
        if $j + 1 == PtrX$ then
            $VP[i-1][j] = random[PtrY - 1]$
        else
            $VP[i-1][j] = random[j]$
    if $PtrX < N_{seg}$ then
        $PtrX++$
    else
        $PtrY++$
        $PtrX = PtrY + 1$
return VP
```

The computed $EncMatrix$ guarantees that in each view generated by the auditor there are $N_{seg-view}$ segments (called utility-preserved segments) encrypted equally (i.e., for the same number of times) using the shared key K_{An}. Thus, the equality property or shared prefixes are only preserved between those segments, and the auditing results of such segments will be valid. In contrast, the results for the remaining segments are fake (i.e., invalid results that look indistinguishable from the valid results). This is meant to prevent the auditor from inferring the virtual topology of the cloud infrastructure as well as the verification results. Furthermore, as the sensitive attributes are initially encrypted with tenants' keys (which are not shared with the auditor), the auditor cannot decrypt the sensitive attributes unless by brute forcing this key (which will be discussed in Chapter 6). The following example shows how the parameters are generated.

Example 3. For a log of five segments ($N_{seg} = 5$) and two real segments per view ($N_{seg-view} = 2$), the CSP computes the following utility preserving parameters:

1. A random vector V_{Random} (kept secret by CSP):

$$V_{Random} = \begin{bmatrix} 3 \\ 5 \\ 2 \\ 7 \\ 4 \end{bmatrix}$$

2. A set of ten vectors $\{VP_i\}_{i<=N_{views}}$ (kept secret by CSP):

$$VP_1 = \begin{bmatrix} \underline{12} \\ \underline{12} \\ 13 \\ 17 \\ 10 \end{bmatrix} ; VP_2 = \begin{bmatrix} \underline{15} \\ 11 \\ \underline{15} \\ 14 \\ 17 \end{bmatrix} ; VP_3 = \begin{bmatrix} \underline{18} \\ 13 \\ 11 \\ \underline{18} \\ 14 \end{bmatrix} ; VP_4 = \begin{bmatrix} \underline{16} \\ 14 \\ 13 \\ 17 \\ \underline{16} \end{bmatrix} ; VP_5 = \begin{bmatrix} 11 \\ \underline{19} \\ \underline{19} \\ 18 \\ 13 \end{bmatrix}$$

$$VP_6 = \begin{bmatrix} 13 \\ \underline{14} \\ 16 \\ \underline{14} \\ 12 \end{bmatrix} ; VP_7 = \begin{bmatrix} 18 \\ \underline{13} \\ 15 \\ 17 \\ \underline{13} \end{bmatrix} ; VP_8 = \begin{bmatrix} 11 \\ 15 \\ \underline{18} \\ \underline{18} \\ 13 \end{bmatrix} ; VP_9 = \begin{bmatrix} 19 \\ 14 \\ \underline{16} \\ 17 \\ \underline{16} \end{bmatrix} ; VP_{10} = \begin{bmatrix} 11 \\ 14 \\ 12 \\ \underline{19} \\ \underline{19} \end{bmatrix}$$

3. An encryption matrix $EncMatrix$ (to be sent to the auditor) calculated as ($EncMatrix[i][j] = VP[i][j] - V_{Random}[j]$), e.g., for the first two elemnts in the first row $9 = 12 - 3$ and $12 = 15 - 3$:

$$EncMatrix = \begin{bmatrix} 9 & 12 & 15 & 13 & 8 & 10 & 15 & 8 & 16 & 8 \\ 7 & 6 & 8 & 9 & 14 & 9 & 8 & 10 & 9 & 9 \\ 11 & 13 & 9 & 14 & 17 & 14 & 13 & 16 & 14 & 10 \\ 10 & 7 & 11 & 10 & 11 & 7 & 10 & 11 & 10 & 12 \\ 6 & 15 & 10 & 12 & 9 & 8 & 9 & 9 & 12 & 15 \end{bmatrix}$$

Anonymization of SegLog. The i^{th} segment in $SegLog$ is then encrypted $V_{Random}[i]$ times using K_{An}. After that, the rows of EncMatrix and the segments of the anonymized $SegLog$ are permuted together randomly to hide the position of the real segments in the generated views. The result of this stage is denoted as $SegLog_p$. Algorithm 2 presents high-level details of this step.

Algorithm 2 UtilityPreservAnonym

```
$V_{Random}$=**GenerateVR ()**
$VP$=**GenerateVP ()**
$K_{An}$=**GenerateKey()**
$EncMatrix$=**GenerateEncMatrix(**$V_{Random}$,$VP_i$**)**          ▷ using Equation 2
$SegLog_e$=**EncryptSegLog(**$SegLog, V_{Random}, K_{An}$**)**
$EncMatrix_p$, $SegLog_p$=**Permute**($EncMatrix, SegLog_e$)          ▷ permutation function
**return** $VP_i$, $K_{An}$, $EncMatrix_p$, $SegLog_p$
```

Example 4. Based on our running example, we show the encryption and permutation steps:

1. The CSP encrypts each segment based on the values in V_{Random} (e.g., segment number 1 will be encrypted 3 times using K_{An}).

2. After that, each row in the $EncMatrix$ is paired with its corresponding segment and randomly permuted (horizontal permutation). The resulting $EncMatrix_p$ and $SegLog_p$ after permutation and encryption are shown in the following:

$$EncMatrix_p = \begin{bmatrix} 11 & 13 & 9 & 14 & 17 & 14 & 13 & 16 & 14 & 10 \\ 10 & 7 & 11 & 10 & 11 & 7 & 10 & 11 & 10 & 12 \\ 7 & 6 & 8 & 9 & 14 & 9 & 8 & 10 & 9 & 9 \\ 6 & 15 & 10 & 12 & 9 & 8 & 9 & 9 & 12 & 15 \\ 9 & 12 & 15 & 13 & 8 & 10 & 15 & 8 & 16 & 8 \end{bmatrix}$$

Tenant-ID	**Net-ID**	**VM-ID**	**VM-Pri-IP**	**VM-Pub-IP**	**Router-ID**	**Next-Hop**
3456	X12	WSA	69.35.7.92	1.10.10.1	WDS	YSQ
8745	12W	W5F	34.16.5.40	1.10.10.4	W5R	RDP
5684	OI1	WE2	75.81.15.34	1.10.10.4	LA4	MVZ
6571	Q4E	WQ1	74.99.41.52	1.10.10.1	W3S	UZQ
9865	IO1	KK2	162.2.10.12	1.10.10.1	QW1	JAQ

Table 6: $SegLog_p$ after permutation.

4.3.4 Multi-View Generation

The auditor takes as input the utility parameters shared by the CSP, namely, $EncMatrix_p$, $SegLog_p$, and K_{An}, and generates the multiple views to be audited as follows. Based on the number of columns in $EncMatrix_p$ (i.e., N_{views}), $SegLog_p$ is cloned into N_{views} copies, called views, and in each view, the total number of segments N_{seg} (i.e., the number of rows of $EncMatrix_p$) and the content of the segments are identified. Then, each view j is encrypted using K_{An} and $EncMatrix_p$, such that each segment i in the view j is encrypted with K_{An} as many times as the corresponding value at (i, j) in the $EncMatrix_p$ (Step 6 in Figure 3).

Example 5. Table 7 and Table 8 respectively show the first and second views generated by the auditor. The shaded segments are those parts that are real such that the equality property and shared prefixes are preserved and their auditing would lead to valid results. The other (unshaded) segments, indistinguishable from the real ones, would lead to fake audit results.

Once all views are generated, the auditor analyzes them (Step 7 in Figure 3), and sends the reports for each view back to the CSP (Step 8 in Figure 3), where each report

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
2478	11E	I3I	63.89.65.15	1.10.10.1	KL6	MZN
5678	4F3	9KL	89.698.45.6	1.10.10.5	OP3	ASP
7231	LX2	WSA	87.65.7.2	1.10.10.1	WDS	GH3
6761	CNN	M8P	36.74.51.46	1.10.10.3	HN3	L4U
7231	NJ2	KFD	87.65.7.65	1.10.10.1	NW1	GH3

Table 7: First view generated by the auditor. Shaded segments are real segments (between which equality or shared prefixes are preserved) and unshaded rows are fake ones.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
5143	23E	2K1	98.62.1.71	1.10.10.5	PLM	GH9
7238	1W2	VCS	14.15.89.54	1.10.10.2	PP9	3KY
9651	DDF	MNI	25.6.99.11	1.10.10.5	HMH	IS1
3456	4TF	WSA	18.2.6.12	1.10.10.1	NF5	T5R
7676	GGF	WQ1	17.2.9.24	1.10.10.1	NJ1	MQF

Table 8: Second view generated by the auditor. Shaded segments are real segments and unshaded rows are fake ones.

is identified using the column index in $EncMatrix$ used to obtain the corresponding view. Note that the security group rules depicted in Figure 1 is encrypted exactly the same way (omitted due to space limitation), which is used together with Table 7 and 8 to generate Figure 4.

Example 6. Figure 4 shows the constructed virtual topologies from the first two generated views in Tables 7 and 8. As we can see, the generated topologies are different from the original topology shown in Figure 1 from several perspectives: e.g., the number of tenants, the number of gateway routers, and virtual infrastructure identifiers. Only a portion of the original topology is preserved in each view (shown with a boldface dashed lines). A malicious auditor trying to recover the virtual network topology, cannot effectively distinguish the real part of the topology from the fake one. Then, the auditor performs reachability verification between all pairs of

VMs in each view (e.g., using NoD [22]). The audit report includes all pairs of tuples (i.e., Tenant-ID, VM-ID, VM-IP, Network-ID, etc.) in the view and their reachability results (reachable/not reachable).

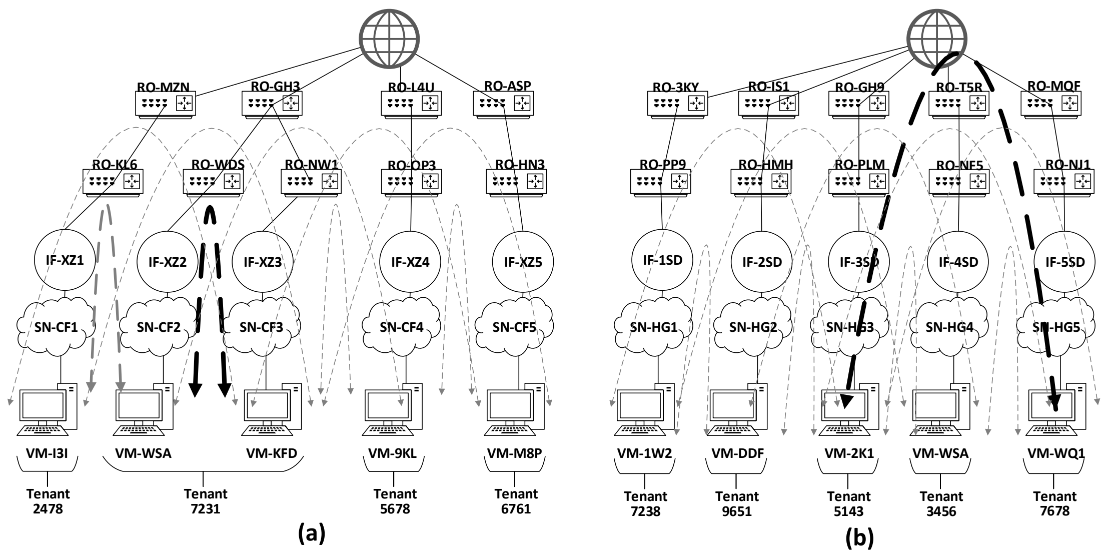


Figure 4: Network topologies inferred from two generated views (together with the encrypted SGR table not shown here): (a) topology related to first view in Table 7, (b) topology related to second view in Table 8. The gray dashed lines are fake and boldface dashed lines are real.

4.3.5 Per-Tenant Report Integration

This step runs at the CSP side and takes as input the reports received from the auditor and their identifiers as described in the previous Chapter, recovers the correct results from all reports for all tenants, and then prepares a per-tenant report (Step 9 in Figure 3). The main operations performed by this step are as follows.

Real Results Extraction and Integration. Report integration (identification of real audit results from fake) can be prepared in advance by CSP. More precisely, for each view i, the secret vector VP_i is used to identify the number of times in total the key K_{An} is used to encrypt each segment of this view. Such that, the positions of equal elements in VP_i with the $SegLog$ records are used to identify the real segments in each view. Thus, view (report) IDs, tuples of information from $SegLog$, the same tuples encrypted using K_{An} and the equal VP_i elements (i.e., underlined elements), are stored locally in the ReportPrep table to be used as follows. Algorithm 3 shows the steps of report integration.

Algorithm 3 ReportsIntegrationPreparation

```
Input: $VP[N_{views}][N_{seg}], SegLog$
Output: ReportPrep
$index \leftarrow \emptyset, val \leftarrow \emptyset, ReportPrep[] \leftarrow \emptyset$
for ($inti = 1; i <= N_{views}; i++$) do
    (index,val)=RepeatedElements(VP[i])          ▷ find real segments and equal VP elements
    RealSeg=ExtractRealSeg(index)        ▷ function to extract segments data assigned index
    ReportPrep[i] = (i,RealSeg,val)
return ReportPrep
```

Example 7. Table 9 shows the example of data extracted and stored to correctly integrate the reports.

Report-ID	Tenant-ID	VM-ID	VM-IP	Enc-Num
1	9998	VVV	66.22.10.3	12
	9998	MX2	66.22.7.66	
2	9998	VVV	18.12.12.14	15
	5554	SQ1	101.2.42.9	

Table 9: Excerpt of ReportPrep prepared by the CSP for the integration of the first two reports.

Per-tenant Report integration and Forwarding. Once all reports are received, the CSP uses the ReportPrep table to search in each report the encrypted results corresponding to the real segments generated by tha auditor and discard the others. Once the results are identified, the CSP replaces the encrypted data in the reports with the data encrypted using tenants' keys K_{Ti} that is the one stored in ReportPrep Table 9, Algorithm 4 implements the preparation for report reception.

Algorithm 4 ReportsIntegration

```
Input: ReportPrep[ ],viewReport[ ]
Output: TenantReports[tenantid][report]
for ($inti = 1; i <= N_{views}; i++$) do
    EncryptedRealSeg=findEncSeg(i,ReportPrep)    ▷ function to find real encrypted segments in view i
    Result=FindInReports($EncryptedRealSeg, viewReport$)    ▷ function to find real results and discard fake
    TenantReports[tenant-id][i]= ReportPerTenant($Result, i$)    ▷ function to store tenants results encrypted using their keys
return TenantReports
```

To avoid any leak from verification results, the CSP forwards per-tenant encrypted report, such that each tenant can only decrypt his/her own data. Meanwhile, all data related to other tenants, are encrypted by other tenants' keys. Thus, *SegGuard* allows each tenant to access the plain IDs and IPs for his/her own resources while s/he is able to have an encrypted evidence about their audit breaches with other tenants, which allows preserving the privacy of sensitive attributes of other tenants. Finally, each tenant T_i decrypts the report using his/her key KT_i (Step 10 in Figure 3).

Example 8. Table 10 shows the final auditing reports forwarded to Tenant1 (*Tenant-ID: 1234*). Table 11 shows the final decrypted auditing reports of Tenant1 (*Tenant-ID: 1234*) using K_{T1}. Based on Table 11, Tenant1 can only see its assets' IDs (e.g., *CD1, CD2, 1234*) and IPs (e.g., *1.10.10.1*) while the identifiers of the other tenant (encrypted *Tenant-ID: 5554*) are still encrypted using his/her key (e.g., *TT2*, and *101.2.42.13*).

Tenant-ID	VM-ID	VM-IP	Tenant-ID	VM-ID	VM-IP	Result
9998	VVV	18.12.12.14	5554	TT2	101.2.42.13	Reachable
9998	MX2	66.22.7.66	9998	GTQ	66.22.17.34	Reachable
9998	GTQ	66.22.17.34	9998	MX2	66.22.7.66	Reachable

Table 10: Final report for Tenant1.

Tenant-ID	**VM-ID**	**VM-IP**	**Tenant-ID**	**VM-ID**	**VM-IP**	**Result**
1234	CD1	1.10.10.1	5554	TT2	101.2.42.13	Reachable
1234	CD2	27.0.2.9	1234	CD3	27.0.3.27	Rechable
1234	CD3	27.0.3.27	1234	CD2	27.0.2.9	Rechable

Table 11: Final decrypted report for Tenant1. Shaded cells are decrypted cells by Tenant1.

Chapter 5

Implementation

In this chapter, we detail the implementation of *SegGuard* and its integration into OpenStack. This chapter also provide an overview of the environment and the tools we use to implement *SegGuard*.

5.1 Background

OpenStack [37] is an open-source cloud infrastructure management platform that uses a set of software tools for managing and building large pools of compute, storage and networking resources. OpenStack is one of the most extensively deployed infrastructure management platforms in today's data centers [17]. The implementation of *SegGuard* mainly involves Nova and Neutron, which are the two main management layer services for the creation and maintenance of virtual infrastructure and networking in the cloud. First, Nova is a compute service that provides tenants on-demand self-service access to compute resources in order to create VMs. Additionally, it allows the creation and maintenance of security groups that play the role of virtual firewalls for the VMs. Second, Neutron is a networking project focused on delivering networking services. This will allow tenants to create and maintain virtual networks between their VMs and connecting their virtual infrastructures to external networks.

5.2 SegGuard Integration into OpenStack

Figure 5 illustrates a high-level architecture of *SegGuard* and shows how it performs the aforementioned steps at three levels: CSP, tenants, and auditor. Our approach interacts with OpenStack services to collect various types of audit data, and with the CSP to obtain the parameters. Specifically, *SegGuard* is integrated into OpenStack by deploying three main components:

1) **Data Collector and Parser Engine**: This component interacts with Nova and Neutron OpenStack components and OpenDaylight controller to retrieve the configuration data stored in databases using SQL queries. For instance, VM ports, router interfaces, router gateways and other virtual ports are collected from table *ports* in Neutron database. Therein, device owner and device ID fields in these tables are used to infer the relationship between the virtual ports and their corresponding devices. We also collect the private and public IPs of VMs from *instances* table, as well as security groups and rules from the *routerrules, subnetroutes and securitygrouprules* tables, where rules are represented by IP destination-nexthop data pairs. Also, OpenDaylight defines a unique flow-ID for each virtual network and maintains its current flow states. The collected data is duplicated in the *SegGuard* database to facilitate more efficient local processing. Then, as the data is scattered over different tables, the engine performs several data pre-processing and filtering steps, such as removing unnecessary data, aggregating relevant data and sorting it based on tenants' identifiers.

2) **Data Anonymizer Engine:** This component performs *SegGuard* operations at the CSP side, where it takes as input the tenants and CSP keys, the number of segments and real segments per view, and data collected and parsed by the data collector

and parser engine as detailed in Chapter 4. The output of this module is the seed log and $EncMatrix_p$.

3) **Anonymization Evaluator Engine:** This component performs two main steps. First, it plays the role of data auditor and takes as input the seed log and the $EncMatrix_p$ to generate the multiple views. Second, the engine evaluates the topology changes between the original data and generated views by the analyst by providing a summary statistics of these differences. This engine is used to evaluate the effectiveness of our approach and build up experiments detailed in Chapter 8.

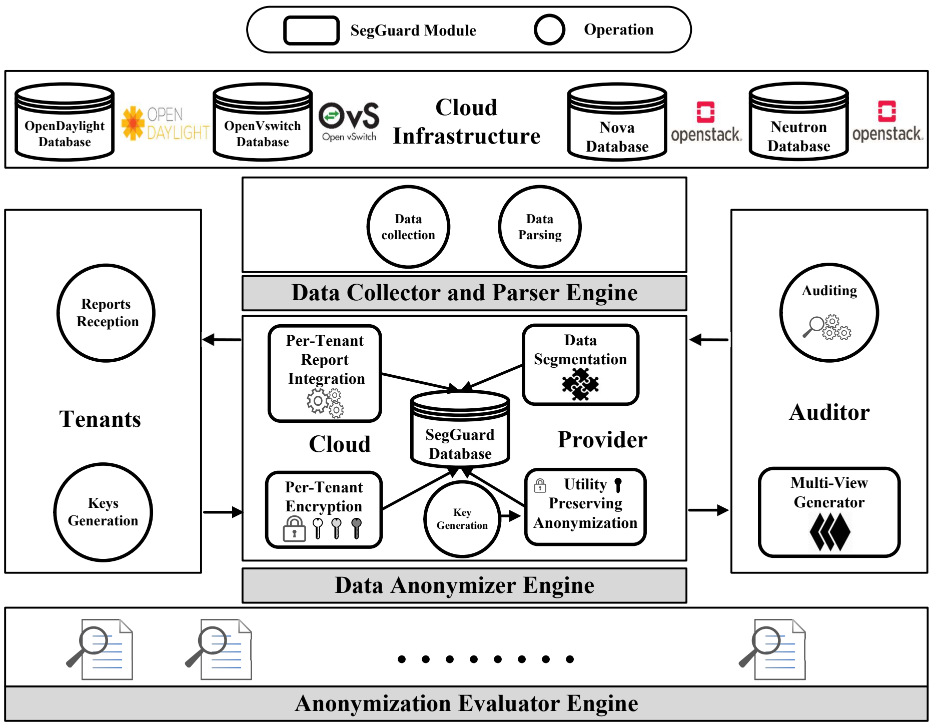


Figure 5: A high-level architecture of *SegGuard*.

5.3 Encryption algorithms

There are two main types of processed data to be anonymized: infrastructure IDs (256 bits) and network components IPs (32 bits). We use two types of data encryption algorithms: To encrypt infrastructure IDs, we use the deterministic Advanced Encryption Standard (AES) [16], and we use the *Prefix-Preserving Anonymization*, influenced by [51], to encrypt the infrastructure IPs, with 256-bit encryption keys for both algorithms. In order to demonstrate the effectiveness, applicability and efficiency of *SegGuard*, we conduct experiments of *SegGuard* based on both synthetic and real data provided by anonymous sources as discussed in Chapter 8.

Chapter 6

Security Analysis

In the following, we first analyze the security and the utility of our solution.

6.1 Security Analysis

We analyze the level of security offered by *SegGuard* to protect the topology information and sensitive attributes, respectively (all the communications are assumed to be over secure channels).

6.1.1 Security of the Topology Information

Generally speaking, existing anonymization techniques would leak important topology information, if we were to apply them directly to our case. For example, directly applying the deterministic encryption and prefix-preserving encryption will preserve both the equality between identifiers and shared prefixes between IPs. As a result, the relationships between virtual resources, e.g., whether two VMs are reachable, will also be preserved in the anonymized audit data. Therefore, the auditor still can potentially construct the real topology of the entire cloud infrastructure based on the anonymized audit data.

In contrast, *SegGuard* provides strong protection of such topology information from the auditor through its segmentation-based, layered encryption technique, as introduced in the previous chapter. Specifically, recall that, as demonstrated in Chapter 5, the seed log does not preserve either the equality property or the shared prefixes, which means the auditor cannot recover any topology information from the seed log. In addition, since the auditor cannot identify the few real segments from a much larger number of fake segments, s/he only has a slim chance to recover such information from the generated views, as will be detailed below and evaluted in the next chapter.

Specifically, as the secret utility parameters are keys to *SegGuard*'s security, we consider a brute force attack performed by an adversary to guess the secret utility parameters in an attempt to recover the utility preserved segments from all views and identify the topology. To this end, the adversary has two choices: to guess either **i)** V_{Random} and use Equation 2 with $EncMatrix$ to compute all VP_i or **ii)** all VP_i in order to recover the original data from all views. In the following, we show that the likelihood of an adversary to succeed in recovering either vector is very low.

The probability of an adversary guessing V_{Random} depends on the size of the domain D for the random number generation (which is only known by the CSP) and the size of the vector, namely, N_{seg}. It can be computed as follows:

$$P\ (V_{Random}) = \frac{1}{D} * \frac{1}{(D-1)} * ... * \frac{1}{D - N_{Seg}} \tag{3}$$

The probability of an adversary guessing one of the VP_i depends on the size of the domain for the random number generation D, the size of the vectors N_{seg}, and the number of views to be generated by the auditor N_{views}, and can be computed as follows:

$$P\ (VP_i) = (\frac{D * 1 * (D-1) * (D-2) * ... * (D-(N_{Seg}-2)}{D^{N_{Seg}}}) * \frac{1}{(N_{Views}! - i)} \tag{4}$$

For instance, for $N_{seg} = 4$ and $D = 200$, the probability to guess the correct V_{Random} is less than $1 * 10^{-9}$. Note that, the probability to guess one VP_i (for example, with $N_{views} = 6$) is even smaller and the adversary needs to recover all of the six VP_is to recover all real segments. Thus, even with a reasonable domain size, it is relatively difficult for an adversary to recover the original topology.

Finally, the $EncMatrix$ is generated randomly using a uniform distribution over X, so the probability to generate any value in the matrix would be $\frac{1}{X}$, which is not considered as information leakage according to the theory of secure multiparty computation [53] since it can be simulated in a polynomial time.

6.1.2 Security of the Sensitives Attributes

To protect the confidentiality of the sensitive attributes, *SegGuard* leverages symmetric key encryption and prefix preserving anonymization techniques in a special manner and rely on the security of the symmetric key encryption algorithms [16]. We use 256-bit encryption keys for both encryption algorithms used in *SegGuard*. Furthermore, it is well known that generally for any key K and plaintext X, it's computationally infeasible to compute K given $(X)_K$. We are assuming that the generated tenants keys and the shared key between CSP and auditor are shared through a secure channel (e.g., publick key infrastructure).

On the other hand, all prefix preserving schemes are subject to a special type of attacks known as the semantic attacks to the same degree. Semantic attacks allow the attackers to infer a prefix or the whole unanonymized addresses by exploiting the cryptanalysis techniques and the prefix preserving semantics. Semantic attacks can be categorized into two forms:

1. Injection attack: In this type of attacks, the adversary injects original data with arbitrary source and destination IP addresses or forging the IP addresses so that they become distinguishable in its anonymized form for later recognition purposes [44]. However, this attack is less likely to be achieved in the cloud infrastructure environment. Whereas, cloud configuration data are based on real cloud operations and tenants can only perform operations based on the capabilities assigned to them by CSP (e.g., create VM, create SGR). So that, a malicious tenant can only inject records into his configuration data (e.g., by generating legitimate operations in his domain within the permissions assigned to him), but cannot inject such data into other tenants' configuration data without compromising the cloud infrastructure.

2. Frequency analysis attack: The frequency analysis attack [7], is performed based on adversary has a certain level of knowledge on the attacked data. This attack returns all possible matches between original and prefix-preserved anonymized IP addresses. It builds a probabilistic model based on the prefixes in data that are within his knowledge and another model for those IPs in the anonymized version of the data, then it starts a matching process between these two models and ends up with a set of deanonymized IP addresses. Moreover, the adversary's knowledge is gained from a carefully designed injection attack [7], inferring the most popular IP addresses with high frequency of occurrence or from adversary's prior knowledge of the traffic distribution [44]. Nevertheless, if the adversary manages to have a certain level of knowledge, namely pre-knowledge, about other tenants' infrastructures (e.g., IP, virtual resources, etc.), our approach hides the real semantic of the configuration data in the anonymized data which

makes launching frequency attacks useless for the attacker. More precisely, *SegGuard* encrypts tenants' data with their respective own keys, distributes the records among different segments, and re-encrypts these segments different times using the key K_{An} such that the IP addresses within the same subnet and located in different segments will have different prefixes. Consequently, the IPs belonging to the same tenants will not share the same prefixes if they are not in the utility preserved segments. In summary, our approach preserves the relationships for each tenant in the utility preserving segments. Meanwhile, in other segments it is not preserved. This is demonstrated in Chapter 8, where we verify the impact of this type of attacks by implementing and testing the frequency analysis attack with some adaptations. We define the information leakage resulting from this attack as follows: *Information Leakage: The number of deanonymized IP addresses after applying the frequency analysis attack on the anonymized log generated by the SegGuard approach over the total number of the original IP addresses.*

6.2 Utility Analysis

Our solution can be shown to be utility preserving based on the following four properties:

1. The use of deterministic encryption guarantees that encrypting the same message with the same initialization vector and key returns always the same ciphertext, for the same tenant. Thus, the same messages before encryption will be mapped to the same messages after encryption (e.g., resources identifiers and their relationships). For example, the tenant-ID allows identifying the owner of a resource. If the same tenant-ID is not mapped consistently, it might affect

the auditing process and cause an incorrect auditing result.

2. The use of per-tenant encryption using tenants specific keys K_{Ti} guarantees that private IP addresses from different tenants will be mapped to different IP addresses to prevent collision. Note that, different tenants can have the same value for different sensitive attributes. For example, the same private IP address *10.5.10.10* in both tenants *T1* and *T2* is converted to two different IP addresses *124.13.10.12* and *73.48.15.12.* after per-tenant encryption.

3. The use of the prefix-preserving anonymization guarantees that IP addresses belong to the same subnet will belong to the same subnet in the anonymized trace. This ensures the utility of data for an audit task that relies on IP addresses and their relationships (e.g., sharing the same prefix). However, the lack of this kind of anonymization results in unusable logs [30, 52]. For instance, the IP addresses of different VMs in the same network should share the same length of prefix after anonymization.

4. The multi-view approach guarantees that the original data can be completely and undistorted audited by verifying the generated views individually. The generated utility preserving parameters in are the key in ensuring this compositionally auditing.

In summary, the aforementioned discussion shows how the utility of the data will be preserved by *SegGuard* in a multi-tenant environment where resources can have the same identifiers/addresses.

6.3 Security of the Communications

All shared keys that involved different parties are assumed to be agreed and distributed among secure channels. Furthermore, the $EncMatrix$ and seed log are transmitted from CSP to the auditor. As discussed earlier, this matrix solely cannot be used to leak any information useful for the adversary. Furthermore, as the equality property and the shared prefixes are both not preserved in the seed log using segmentation and layered encryption, an adversary cannot infer any information about neither the attributes nor their relationships. Additionally, the reports generated by the auditors are encrypted using the secret key K_{An} agreed between the auditor and the CSP to prevent any MiTM. Furthermore, the report integration module provides a second layer of protection as it generates per-tenant reports to prevent a tenant obtains a copy of other tenants' auditing results. Additionally, all sensitive attributes in the retrieved reports are encrypted either using the key of the receiving tenant or using the other tenants' keys to prevent information leakage across tenants. The per-tenant encryption ensures that if all reports fall within adversary hands, they cannot disclose any information related to the sensitive attributes and their respective tenants.

Chapter 7

Discussions

This chapter discusses several aspects of *SegGuard*.

7.1 Computational Cost and Benign Auditor

By preserving data utility, our solution essentially imposes a tradeoff between security and computational cost. Specifically, the selection of the utility parameters (i.e., N_{seg} and $N_{seg-view}$) in Section 4.3.2 determines the level of security and privacy protection provided by our approach, as well as its computational cost. These two parameters determine the number of views to be generated and audited by the auditor. A larger number of views certainly implies a higher level of security and privacy but a higher computational cost as well. In practice, the CSP can select the values of these parameters based on the security and privacy requirements and the amount of computation cost s/he can afford. If the CSP has ahigher level of trust in a given auditor, e.g., its own employees such as cloud administrators, s/he can reduce the computation cost by decreasing the number of views to be generated. To do so, CSP can either decrease the number of utility segments per view or choose a small number of segments in the first place. In contrast, for less trusted third-party auditors, the

CSP can increase those parameters to improve the security.

7.2 Colluding Adversaries

If two or more adversary tenants collaborate to infer information about other benign tenants, the CSP can react in two ways. First, if two or more tenants choose the same key accidentally or intentionally, the CSP should detect and revoke those keys, since failing to do so would allow a higher proportion of its topology to be disclosed to those adversaries. Second, if a tenant and an auditor collude, it would be equivalent to an adversary having more prior knowledge about the data (i.e., the data of the colluding tenant) launching the frequency analysis attack. As we have shown through experiments, *SegGuard* is robust against such adversaries as we will discuss in the experimental results chapter 8.

7.3 Auditing Static Cloud Configurations

SegGuard supports auditing on cloud configurations collected as snapshots through our data collection engine (as described in Chapter 5). Through our experimental results (in Chapter 8), we show that our approach is practical for auditing large-sized clouds, e.g., 62 seconds to anonymize data related to 26 thousand VMs. However, an incremental approach of *SegGuard* will allow collecting, processing and auditing a small amount of configuration changes more efficiently, which we consider as an interesting future work.

7.4 Communication and Storage Costs

SegGuard supports both third party and in-house auditors. In the third party auditing case, transmitting the anonymized audit data and other necessary parameters may involve certain communication cost. However, note that our design of the seed log means only one copy of the audit data will need to be sent over the network in such a case, even though the auditor will need to generate multiple views later on his/her site. On the other hand, if the auditor is a cloud administrator or the third party auditor remotely logins to perform in-cloud auditing, then no data would need to be sent over the network. In both cases, the storage cost for storing the multiple views could be a concern especially when the number of views is large. One easy way to mitigate this issue is to generate, audit, and then delete each view before generating the next view, assuming the auditing results have lower storage requirements.

7.5 Formal Treatment and Applicability on Different Attributes

This work mainly leverages existing crypto primitives such as PPE, permutation and RNG, which allows preserving the prefixes and equality properties for utility purposes. Consequently, the security of our solution is depending on those techniques. These primitives are known to have some weaknesses, but we overcome their weaknesses using the segmentation, where there is a partial preservation of these properties in each view (each view mixed with real and fake segments). We have focused on the system aspect of the solution rather on the formal analysis of the security. That is said, we have performed an informal analysis in the security analysis Chapter 6 and the experimental results Chapter 8, where we measured the information leakage caused by attacks against PPE such as fingerprinting and injection that we implemented and run over our datasets. Our solution can employ any PPE techniques (equality-preserving, prefix-preserving, order-preserving, format preserving, etc.) on any data attribute (e.g., Timestamp, string, integer) with little modification. As such, the IP addresses and the identifiers (e.g., tenant id, network id) are only meant as examples of property preservation encryption (PPE) techniques.

7.6 Use-Cases of SegGuard

We believe that our solution can be applied to any system has different actors with different trust level. For example, in the Central Authentication Services (CAS) applications which involve three parties: client browser, web application and CAS server that has a DB server which communicates with the web application through it. Another use-case could the foreign network firewalls where roaming users are using encrypted channels to protect the security and privacy of their communication, but their data cannot be examined and regulated by foreign networks firewalls. This problem can be relieved if users reveal their data or if the foreign networks reveal their firewalls rules to the tunnels endpoints. With these conflicting security and privacy requirements, it is very difficult to regulate the encrypted tunnels using conventional firewall techniques, because they all require a single entity to possess knowledge on both the connection characteristics and the firewall rules. However, using our solution, the firewall will play the rule of the auditor and the users can play the rule of the CSP and the tenants together. Furthermore, our solution can be applied to encrypted and secure storage DB in order to hide the hierarchy and relationships between data attributes in a database storage system. Finally, we will leave the discussions of these use-cases as a future work, whereas they require further security analysis and experimental investigation.

Chapter 8

Experiments

This chapter presents experimental results measuring the effectiveness and efficiency of *SegGuard*.

8.1 Experimental Setup

In the following experiments, we use both real and synthetic data. Real data consists of two different datasets provided by two major telecommunication companies. The first dataset contains 10K distinct IP addresses, and is used to validate our approach against the frequency analysis attack. The second one is virtual network configuration data from a real cloud environment with 22 physical machines, 37 tenants and 4,377 VMs, and used to study the applicability of our approach as discussed. Furthermore, we used a synthetic dataset collected from our testbed deployment of OpenStack (version Kilo) cloud environment in order to simulate a large cloud deployment of 1.2K virtual routers, 3.2K subnets, 25.2K different VMs and 43K IP addresses.[1] Finally, our scripts for the experiments implemented in C++ and bash were run on a PC with Debian 13.6 64-bits, Intel Core i7 CPU, and 16GB memory.

[1]This is considered a large cloud according to [39] as 94% of OpenStack deployments have less than 10K distinct IP addresses.

8.2 Information Leakage under Semantic Attacks

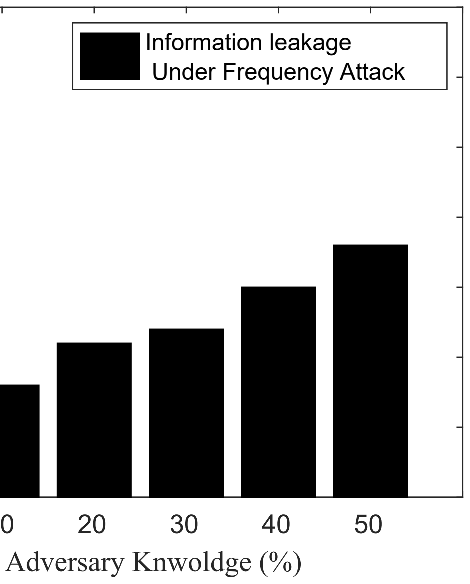
In this chapter, we evaluate the effectiveness of *SegGuard* by examining the *information leakage* (Chapter 2) of our solution under the *Frequency Analysis Attack* defined in Chapter 6, based on the data generated by both the CSP (seed log) and the auditor (all views), while varying four main parameters: the number of segments (N_{seg}), the number of real segments per view ($N_{seg-view}$), the adversary knowledge and the number of VMs deployed in the cloud. In the following, we discuss the obtained results in detail.

8.2.1 Impact of Varying the Number of Segments

In this experiment, we analyze the information leakage for a 10K VMs log against an adversary with 25% knowledge of the original data while varying the number of segments. Figure 6 shows that the information leakage decreases abruptly when the number of segments increases. As expected, while increasing the number of segments, the IP addresses with the same prefixes are distributed further among these segments, which causes the information leakage to decrease. Indeed, applying different layers of anonymization for different segments results in IP addresses that were originally sharing the same prefix to have different prefixes. Thus, when the adversary calculates the frequencies of the original log based on his/her pre-knowledge, s/he is not able to find them on the anonymized version of the log.

8.2.2 Impact of Varying the Number of Real Segments/View

In this experiment, we analyze the information leakage for 10 segments and 25% of adversary knowledge while varying the number of real segment per view. Figure 7 shows that the information leakage increases steadily when the number of real (utility



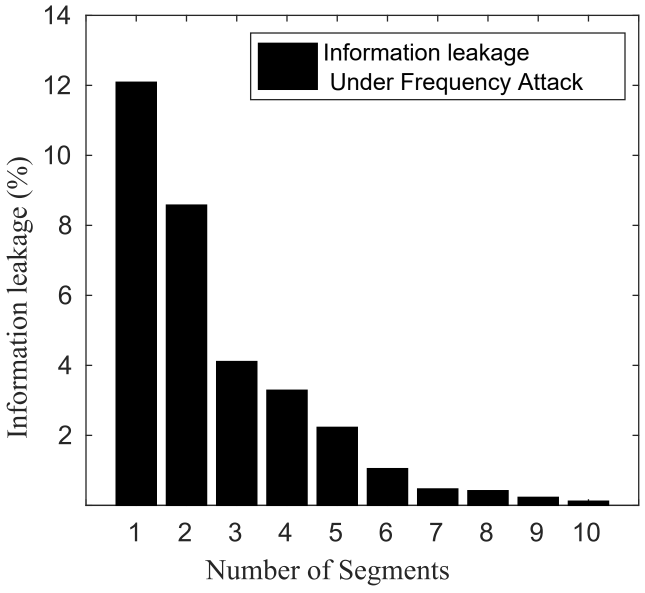
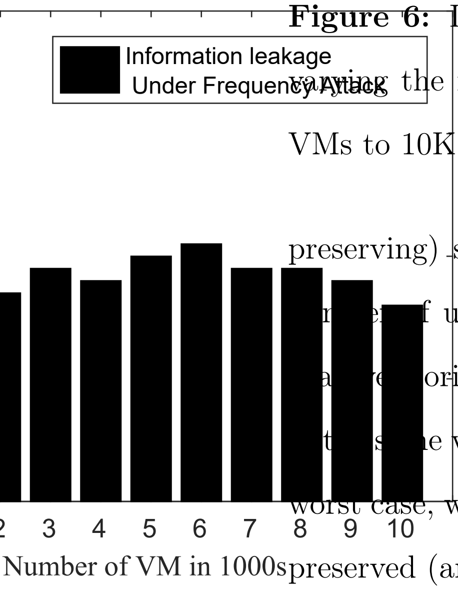
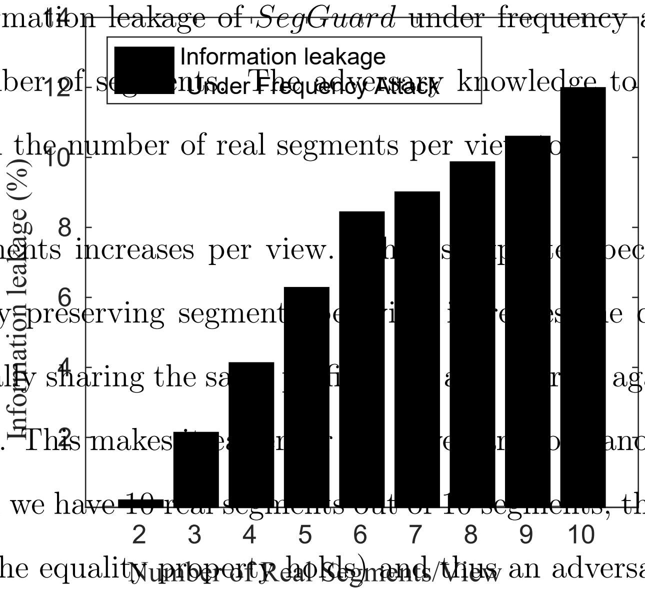


Figure 6: Information leakage of *SegGuard* under frequency analysis attack while varying the number of segments. The adversary knowledge to 25%, the number of VMs to 10K and the number of real segments per view to [illegible]





preserving) segments increases per view. [illegible] because, having a large [illegible] utility preserving segment[illegible] increases [illegible] chances that the IPs [illegible] originally sharing the sa[illegible] again the same prefixes [illegible] view. This makes [illegible] anonymize them. In the worst case, when we have 10 real segments out of 10 segments, the whole log is prefix-preserved (and the equality property holds) and thus an adversary can deanonymize 1.2K of the IPs, which represents 12% of the whole log. However, when the number of real segments is only two, the adversary can only deanonymize 22 IPs from the whole log, which represents 0.22%. Note that this leakage percentage is computed over all IPs of all tenants in the data center that are within the adversary knowledge. Our data segmentation approach ensures that the utility preserving segments related to a single tenant are spread over multiple views. Thus, relatively to a given tenant, this percentage is even smaller. We also note as discussed in Chapter 7, for a given

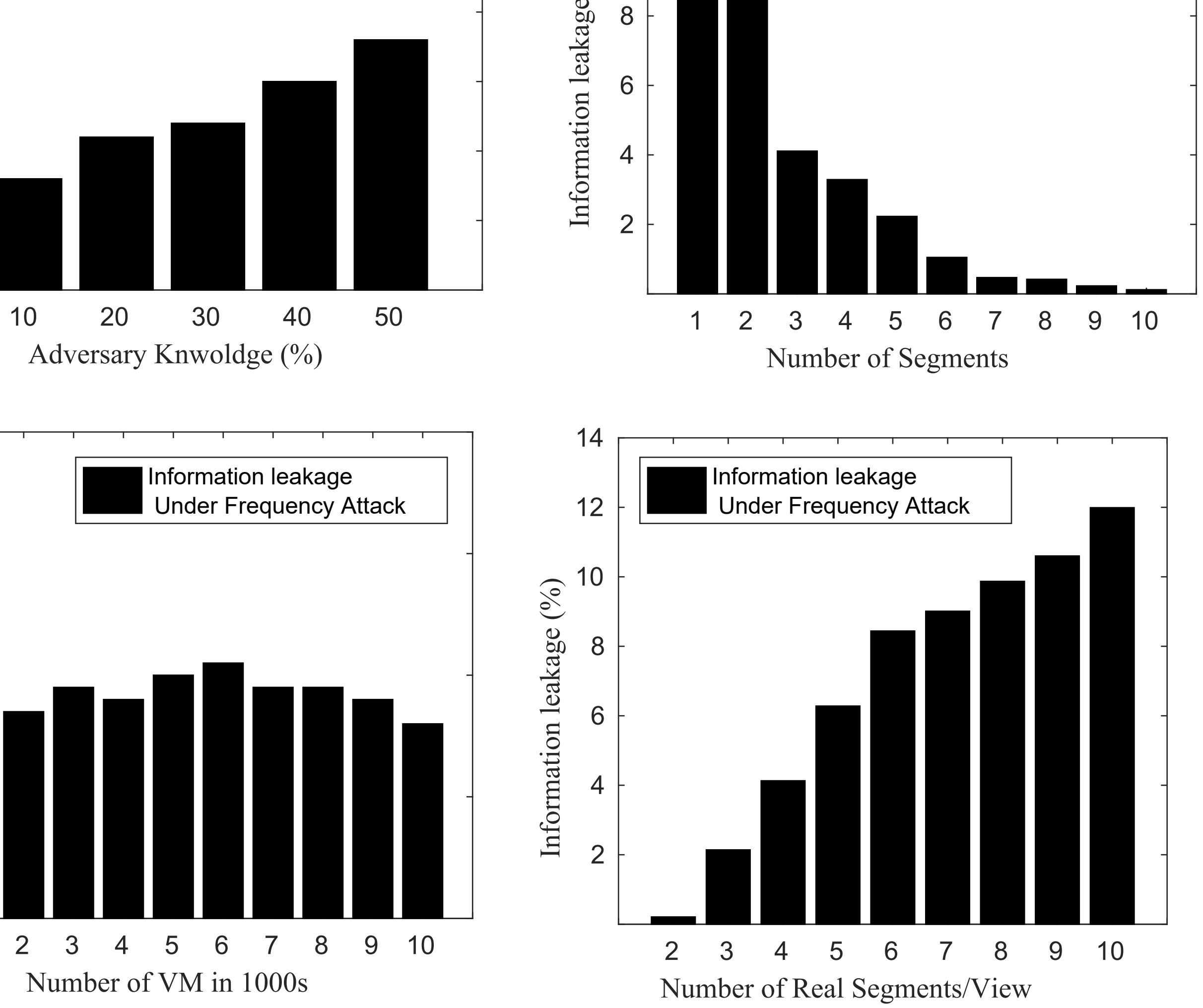
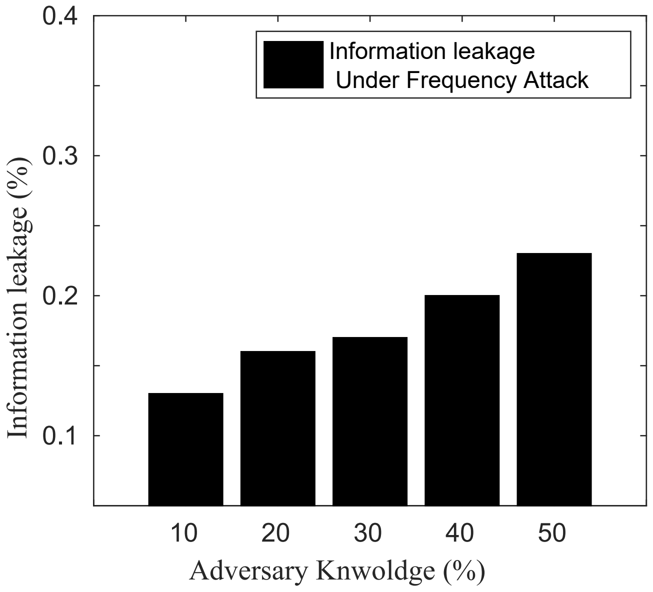


Figure 7: Information leakage of *SegGuard* under frequency analysis attack while varying the number of real segments/view. The adversary knowledge to 25%, the number of VMs to 10K, the number of segments to 10.

8.2.3 Impact of Varying the Adversary Knowledge

In this experiment, we analyze the information leakage whan we have 10 segments and two real segments per view, while varying the adversary knowledge from 10% to 50% of the 10K VMs. Figure 8 shows the information leakage increases slightly with the adversary knowledge (e.g., when adversary knowledge increases from 10% to 50%, the information leakage increases by 0.1%) but stays under a maximum of 0.23%. Such a small percentage of information leakage results from the data segmentation which parcels the data records with the same IP prefixes over several segments, and thus make same IP sharing the same prefixes to be encrypted with different number of layered encryption. Therefoe, when an adversary builds the probabilistic model for



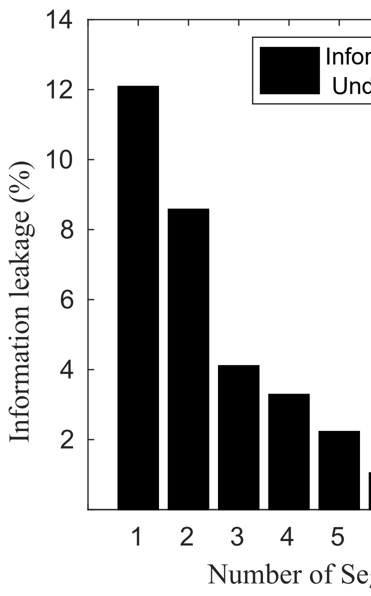
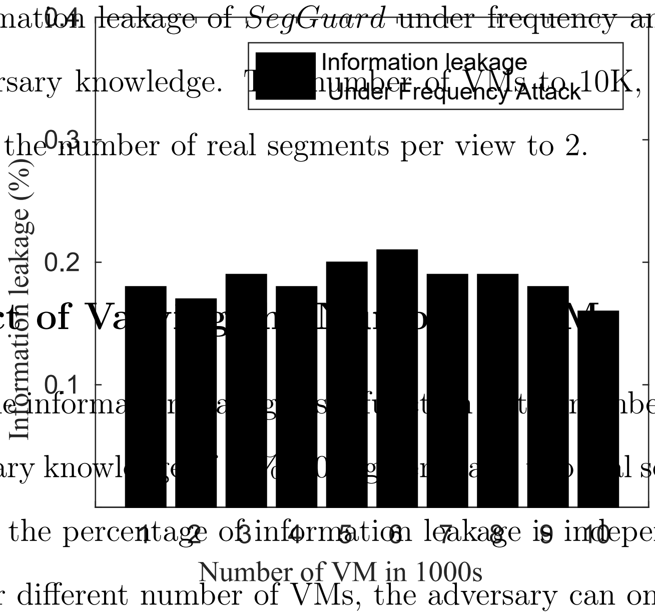
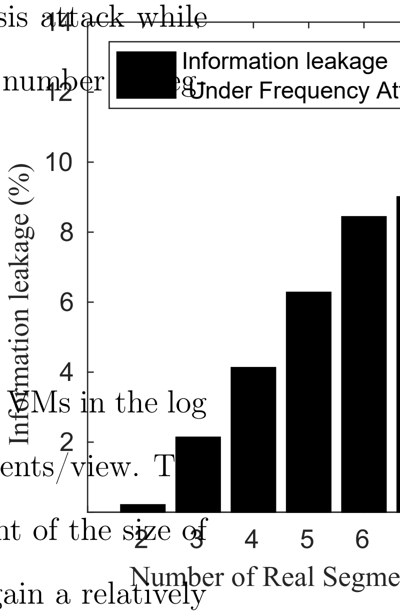


Figure 8: Information leakage of *SegGuard* under frequency analysis attack while varying the adversary knowledge. T[illegible] number of VMs to 10K, the number [illegible] ments to 10, and the number of real segments per view to 2.





8.2.4 Impact of Va[illegible]

Figure 9 shows the informa[illegible] ber of VMs in the log for a fixed adversary knowl[illegible] segments/view. T[illegible] figure shows that the percentage of information leakage is independent of the size of the log. Thus, for different number of VMs, the adversary can only gain a relatively small amount of knowledge. For instance, for the largest dataset, the information leakage is 0.20%. The maximum information leakage is 0.21% that is achieved for a

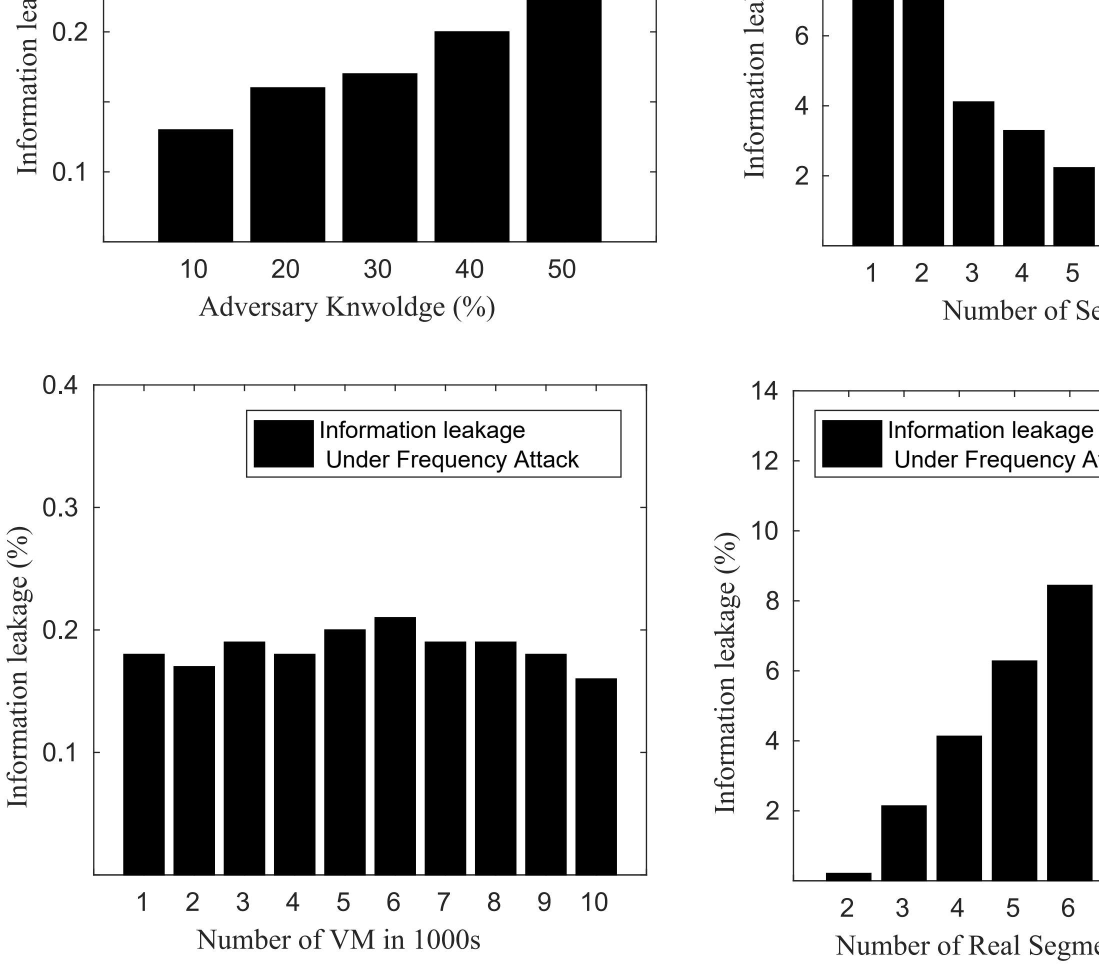


Figure 9: Information leakage of *SegGuard* under frequency analysis attack while varying the number of segments. The adversary knowledge to 25%, the number of segments to 10, and the number of real segments per view to 2.

8.3 Topology Preservation in Real Cloud

For this experiment, we used the dataset provided from a real cloud data center to show the applicability of *SegGuard* to anonymize the network topology in real cloud deployment. Particularly, we show how these relationships change between different views generated by *SegGuard*. Figure 10.(A) illustrates the actual structure of this data center, where it is composed of two racks, $Rack_1$ and $Rack_2$, connected to two edge switches, namely Edg_11 and Edg_12, which are connected to two aggregate switches, namely *Agg11* and *Agg12*. Each rack consists of 11 physical servers and hosts assets from 37 tenants.

We anonymize the configuration data corresponding to this setup using *SegGuard*, where we set the number of segments to four and the number of real segments per view to two. After that, we selected two random views generated by the auditor and visualized their corresponding topology as illustrated in Figure 10.(B) and Figure 10.(C). As we can see, the total number of tenants in the former figure is 131 while in latter it is 107, which are different from the real settings. This is due to the fact that since the tenants' identifiers are spread over the four segments and encrypted different number of times using the key. Particularly, the same tenant identifier will be mapped to different values depending on the segment index and the generated view. Thus, it will appear as a different tenant in each view. The same scenario happens with the identifiers of the virtual resources, the physical servers, the racks, as well as the edge and aggregation switches.

8.4 Efficiency of SegGuard

To measure the performance of our approach, we run our approach on a large dataset of 25.2K VMs and their associated security group policies and measured the time as well as memory and CPU consumptions while varying the number of segments and the range of random values, respectively.

8.4.1 Time Consumption

Figure 11 shows that the time required to prepare the seed log linearly increases with the number of segments. The time difference between having one segment and 20 segments is about 36 seconds.

Figure 12 shows the time consumption when varying the domain of the random generator used to set the values of the vector V_{Random} and the set of vectors VP_i.

It shows that by increasing the random number domain, the required time increases exponentially. However, the security-level offered by our solution is considered sufficient even for a reasonably chosen random domain size. For instance, the required time to generate the seed log by the CSP is about 62 seconds for a domain of 10^2. For the adversary, if the random number domain is 200 and the $N_{Seg} = 3$, then based on Equation 3 there are $(200 * 199 * 198)$ candidates vectors V_{Random} that the adversary has to consider to deanonymize the log. This roughly translates into 15.5 years for the adversary to find the correct $VRandom$, with a machine having the same specification as the one described in the experimental setup. Note that, we consider the computation consumption at the cloud provider as the latter is more concerned with the amount of resources s/he needs to put in place for the anonymization process before actually transmitting the information to the analyst. We consider the more detailed evaluation of this issue as future work.

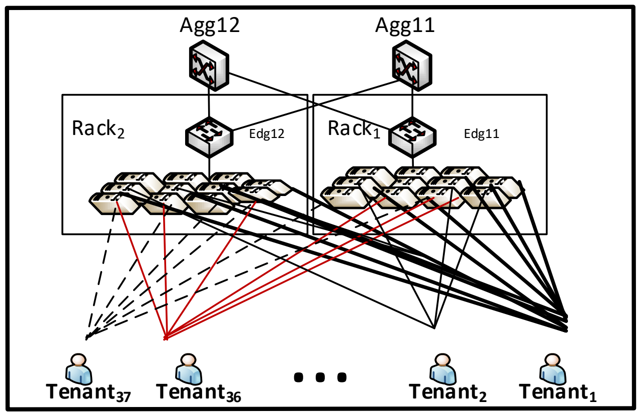
8.4.2 Memory and CPU Consumption

Figure 13 shows that the CPU consumption increases almost linearly from 32.6% to 34.93%, while varying the number of segments from 1 to 20, due to the encryption operations applied to each segment based on the value of V_{Random}.

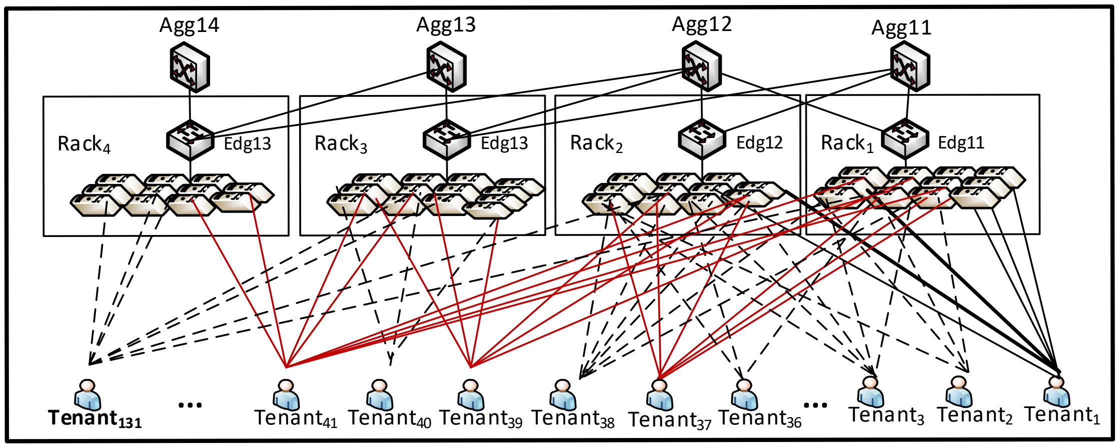
Figure 14 reports memory consumption, the slope of the memory consumption increases linearly with the number of segments due to the multiple read/write operations on each segment to encrypt its records. However, it reaches 3.93% when the number of segments is equal 20. This shows the efficiency of our solution from CPU and memory consumptions perspectives.

8.4.3 Results Summary

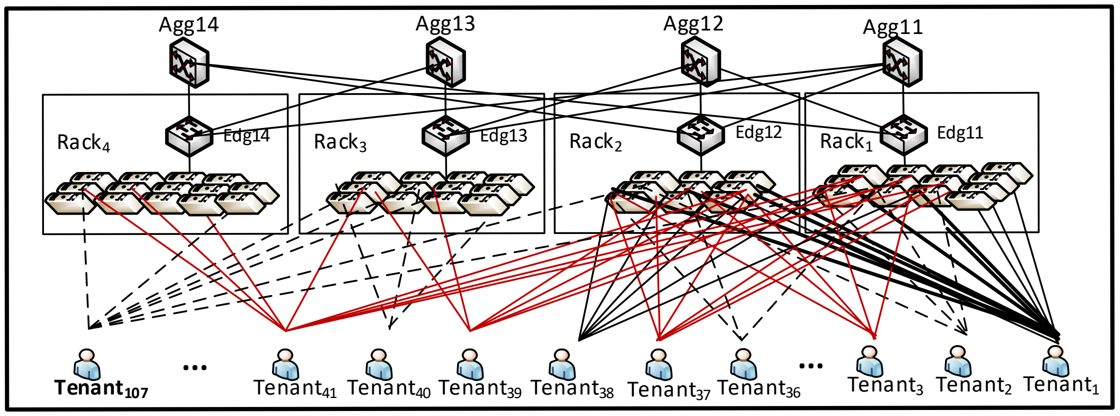
In summary, our experiments show the efficiency of our solution in terms of computational cost. *SegGuard* can anonymize data belonging to a large-scale cloud environment in 62 seconds with a high-level of security and privacy protection guarantees.



(A)



(B)



(C)

Figure 10: Network topology corresponds to two different views selected randomly from the views generated by *SegGuard*; where we set the N_{seg} to 4 and the $N_{seg-view}$ to 2. The total number of tenants in figure B is 131 while in C it is 107, which are different from the real settings.

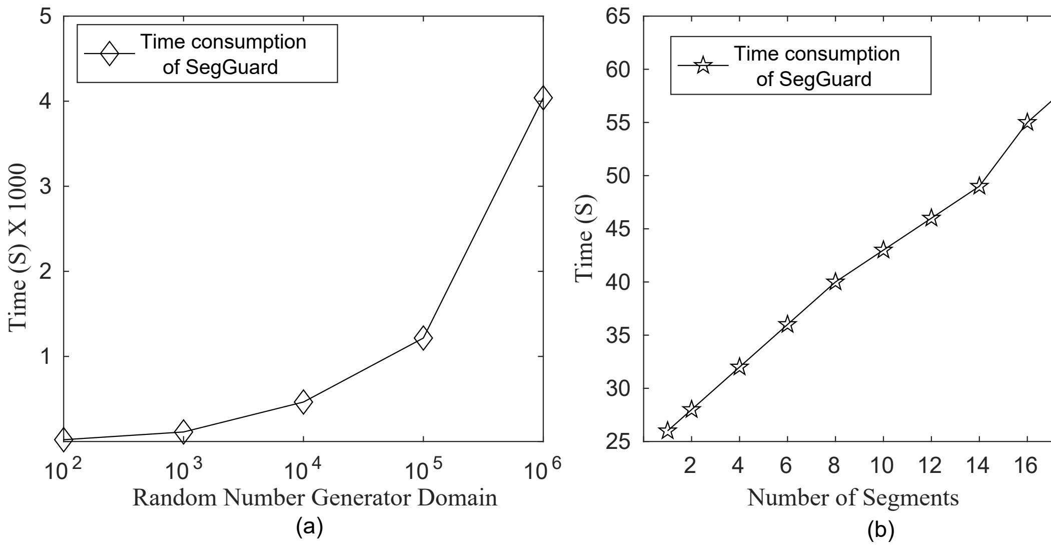
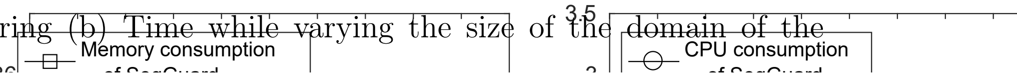


Figure 11: Efficiency of *SegGuard*: Measuring (a) Time while varying the number of segments and measuring (b) Time while varying the size of the domain of the



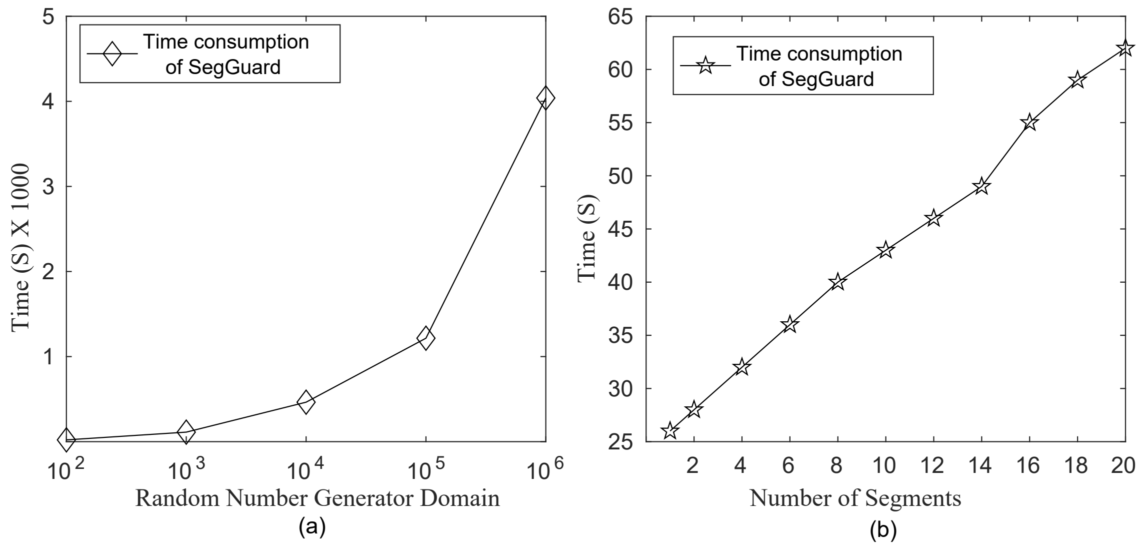
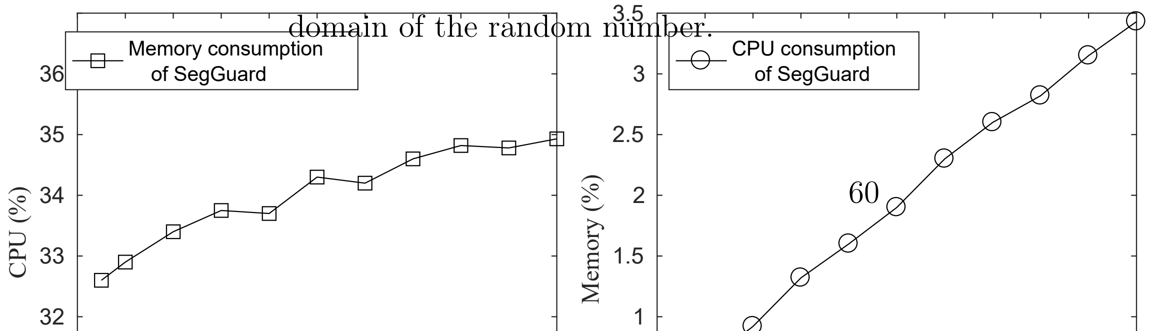
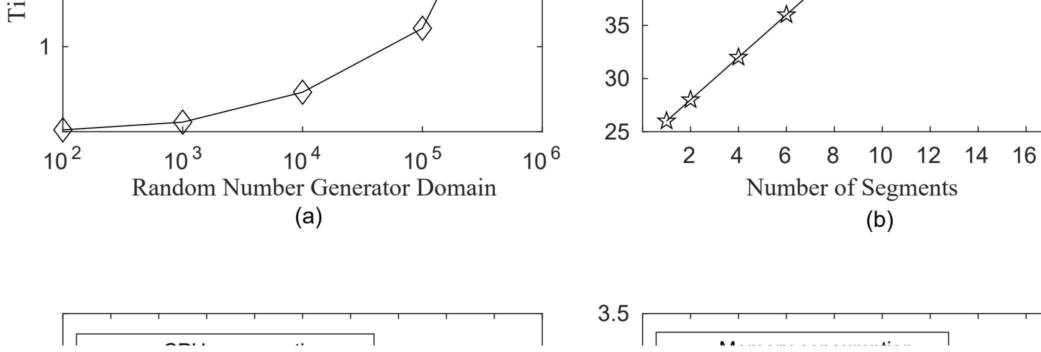


Figure 12: Efficiency of *SegGuard*: Measuring time while varying the size of the domain of the random number.





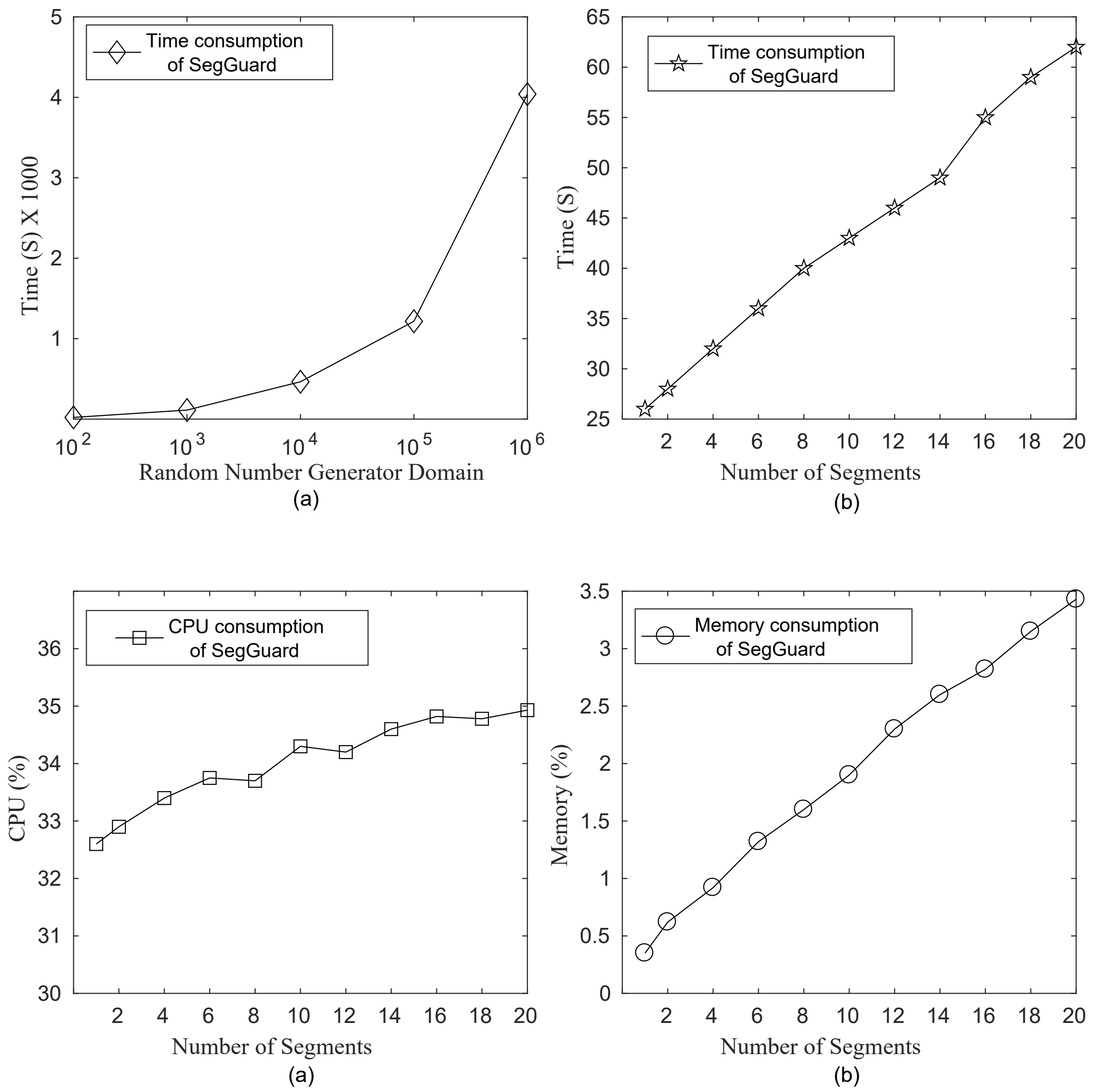


Figure 14: Efficiency of *SegGuard*: Measuring Memory consumption while varying the number of segments.

Chapter 9

Conclusion

In this thesis, we presented a novel anonymization approach that limited the leakage of sensitive and private information from either the audit data or audit results in a multi-tenancy cloud, while preserving sufficient utility for effective security auditing. Our solution, *SegGuard*, could allow cloud server provider and tenants to benefit from security auditing done by a semi-trusted auditor without leaking sensitive information about the Cloud server provider, including topology information, to the auditor, or leaking such information across tenants. We evaluated both the effectiveness and efficiency of our solution. Also, we showed the efficiency of our approach since *SegGuard* can anonymize data consisting of 25.2K VMs in an acceptable time as well as memory and CPU consumptions are acceptable. The main limitations of our work and corresponding future directions are as follows. First, we do not consider the integrity of audit data and audit results, which could potentially be addressed with existing integrity mechanisms; we also fully rely on the CSP and assume an honest but curious adversary model, so to what extent can those assumptions be weakened will need to be studied. Second, we have mainly focused on prefix preserving encryption, and a future direction is to expand our scope by incorporating other property-preserving anonymization techniques, and also by studying whether the segmentation-based approach can even be used to hide original data to facilitate auditing tasks that require such data (e.g., the payload). Third, to preserve data utility, our approach basically imposes a trade-off between privacy and computational cost, so optimizing such a trade-off is also an interesting future topic.

Bibliography

[1] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. In *CIDR*. Citeseer, 2013.

[2] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*, pages 535–552. Springer, 2007.

[3] S. Bleikertz, T. Groß, M. Schunter, and K. Eriksson. Automated information flow analysis of virtualized infrastructures. *Computer Security–ESORICS 2011*, pages 392–415, 2011.

[4] S. Bleikertz, C. Vogel, and T. Groß. Cloud Radar: near real-time detection of security failures in dynamic virtualized infrastructures. In *Proceedings of the 30th annual computer security applications conference*. ACM, 2014.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 224–241. Springer, 2009.

[6] T. Brekne and A. Årnes. Circumventing ip-address pseudonymization. In *Communications and Computer Networks*, pages 43–48, 2005.

[7] T. Brekne, A. Årnes, and A. Øslebø. Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *PET*, pages 179–196. Springer, 2005.

[8] M. Broy, S. Chakraborty, D. Goswami, S. Ramesh, M. Satpathy, S. Resmerita, and W. Pree. Cross-layer analysis, testing and verification of automotive control software. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 263–272. ACM, 2011.

[9] M. Burkhart, D. Brauckhoff, M. May, and E. Boschi. The risk-utility tradeoff for ip address truncation. In *NDA*, pages 23–30. ACM, 2008.

[10] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy. Self-service cloud computing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 253–264. ACM, 2012.

[11] P. Carey. *Data protection: a practical guide to UK and EU law*. Oxford University Press, Inc., 2009.

[12] F. Chen, B. Bruhadeshwar, and A. X. Liu. Privacy-preserving cross-domain network reachability quantification. In *ICNP*, pages 155–164. IEEE, 2011.

[13] T.-S. Chou. Security threats on cloud computing vulnerabilities. *International Journal of Computer Science & Information Technology*, 5(3), 2013.

[14] V. Ciriani, S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *ESORICS*, pages 171–186. Springer, 2007.

[15] S. E. Coull, F. Monrose, M. K. Reiter, and M. Bailey. The challenges of effectively anonymizing network data. In *CATCH*. IEEE, 2009.

[16] J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[17] Data center knowldge. One-third of cloud users clouds are private, 2015. Available at: `http://www.datacenterknowledge.com`.

[18] W.-P. De Roever. *Concurrency Verification: Introduction to Compositional and Non-compositional Methods*, volume 54. Cambridge University Press, 2001.

[19] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[20] G. T. Duncan, S. E. Fienberg, R. Krishnan, R. Pad-man, and S. F. Roehrig. Disclosure limitation methods and information loss for tabular data. *Confidentiality, Disclosure and Data Access: Theory and Practical Appli-cations for Statistical Agencies*, pages 135–166, 2001.

[21] J. Fan, J. Xu, and M. H. Ammar. Crypto-pan: Cryptography-based prefix-preserving anonymization. *Computer Networks*, 46(2), 2004.

[22] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. D. Millstein. A general approach to network configuration analysis. In *NSDI*, pages 469–483, 2015.

[23] O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[24] I. Gul, A. ur Rehman, and M. H. Islam. Cloud computing security auditing. In *ICNIT*, pages 143–148. IEEE, 2011.

[25] L. A. Gunawan and P. Herrmann. Compositional verification of application-level security properties. In *ESSoS*, pages 75–90. Springer, 2013.

[26] Y. Han, J. Chan, T. Alpcan, and C. Leckie. Using virtual machine allocation policies to defend against co-resident attacks in cloud computing. *IEEE Transactions on Dependable and Secure Computing*, 14(1):95–108, 2017.

[27] J. Katz and Y. Lindell. Introduction to modern cryptography: principles and protocols. cryptography and network security, 2008.

[28] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *NSDI*, pages 99–111, 2013.

[29] L. Kleinrock. *Queueing systems, volume 2: Computer applications*, volume 66. wiley New York, 1976.

[30] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. *ACM SIGCOMM Computer Communication Review*, 30(4), 2000.

[31] H. Liu. A new form of dos attack in a cloud and its avoidance mechanism. In *CCSW*, pages 65–76. ACM, 2010.

[32] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *NSDI*, 2015.

[33] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Ven-kitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, pages 24–24. IEEE, 2006.

[34] F. McSherry and R. Mahajan. Differentially-private network trace analysis. *SIGCOMM Comput. Commun. Rev.*, 40(4):123–134, Aug. 2010.

[35] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[36] P. Nikander. Ip address ownership verification mechanism, Dec. 26 2006. US Patent 7,155,500.

[37] OpenStack oraganization. Openstack, 2017. Available at: `https://www.openstack.org/`.

[38] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, 2006.

[39] D. Riboni, A. Villani, D. Vitali, C. Bettini, and L. V. Mancini. Obfuscation of sensitive data in network flows. In *INFOCOM*, 2012.

[40] D. Riboni, A. Villani, D. Vitali, C. Bettini, and L. V. Mancini. Obfuscation of sensitive data for incremental release of network flows. *IEEE/ACM Trans. Netw.*, 23(2):672–686, Apr. 2015.

[41] P. Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[42] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its en-forcement through generalization and suppression. Technical report, SRI International, 1998.

[43] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *USENIX security symposium*, pages 175–188, 2012.

[44] A. Slagell and W. Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. In *SECURECOMM*, pages 80–89. IEEE, 2005.

[45] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *CCS*, pages 401–412. ACM, 2011.

[46] T. M. Truta and B. Vinay. Privacy protection: p-sensitive k-anonymity property. In *ICDEW*, pages 94–94. IEEE, 2006.

[47] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Infocom, 2010 proceedings ieee*, pages 1–9. Ieee, 2010.

[48] Y. Wang, T. Madi, S. Majumdar, Y. Jarraya, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. Tenantguard: Scalable runtime verification of cloud-wide vm-level network isolation. In *NDSS*, 2017.

[49] A. Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6):62–67, 2004.

[50] X. Xiao and Y. Tao. Personalized privacy preservation. In *SIGMOD/PODS*, pages 229–240. ACM, 2006.

[51] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix-preserving ip traffic trace anony-mization. In *IMW*, 2001.

[52] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *ICNP*, pages 280–289. IEEE, 2002.

[53] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[54] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat. Libra: Divide and conquer to verify forwarding tables in huge networks. In *NSDI*, volume 14, pages 87–99, 2014.

[55] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *SOSP*, pages 203–216. ACM, 2011.