SegGuard: Segmentation-based Anonymization of Network Data in Clouds for Privacy-Preserving Security Auditing

Momen Oqaily, Concordia University, Yosr Jarraya, Ericsson Security Research,
Meisam Mohammady, Concordia University, Suryadipta Majumdar, University at Albany-SUNY,
Makan Pourzandi, Ericsson Security Research, Lingyu Wang, Concordia University,
and Mourad Debbabi, Concordia University

Abstract—Security auditing allows cloud tenants to verify the compliance of cloud infrastructure with respect to desirable security properties, e.g., whether a tenant's virtual network is properly isolated from other tenants' networks. However, the input to such an auditing task, such as the detailed topology of the underlying cloud infrastructure, typically contains sensitive information which a cloud provider may be reluctant to hand over to a third party auditor. Additionally, auditing results intended for one tenant may inadvertently reveal private information about other tenants, e.g., another tenant's VM is reachable due to a misconfiguration. How to anonymize both the input data and the auditing results in order to prevent such information leakage is a novel challenge that has received little attention. Directly applying most of the existing anonymization techniques to such a context would either lead to insufficient protection or render the data unsuitable for auditing. In this paper, we propose *SegGuard*, a novel anonymization approach that prevents cross-tenant information leakage through per-tenant encryption, and prevents information leakage to auditors through hiding real input segments among fake ones; in addition, applying property-preserving encryption in an innovative way enables *SegGuard* to preserve the data utility for auditing while mitigating semantic attacks. We implement *SegGuard* based on OpenStack, and evaluate its effectiveness and overhead using both synthetic and real data. Our experimental results demonstrate that *SegGuard* can reduce the 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 infrastructure with 25,000 virtual machines).

Index Terms—Privacy preserving, Cloud computing security auditing, Data anonymization.

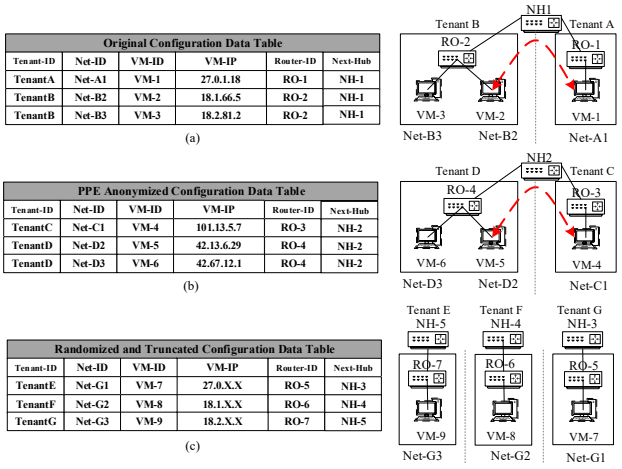
1 Introduction

The lack of transparency in clouds renders security auditing a desirable capability for both the cloud service provider (CSP) and the cloud tenants. A CSP may perform security auditing on its cloud infrastructure, either in-house by a security admin, or by an appointed third-party auditor, in order to detect misconfigurations and vulnerabilities. The auditing also allows the CSP to demonstrate the security compliance of its cloud infrastructure, either as part of the service level agreements, or with respect to security standards (e.g., ISO 27017 [1] and CCM 3.0.1 [2]). A cloud tenant may also perform security auditing on its own virtual infrastructure to ensure desirable security properties are satisfied, e.g., network isolation (i.e., the tenant's virtual network is properly isolated from other tenants' networks), which is among the foremost security concerns in multi-tenant clouds [3].

On the other hand, security auditing in a multi-tenant cloud may present unique security and privacy challenges. First, an auditing task, such as verifying network isolation, may involve a large amount of sensitive input data about both the physical and virtual infrastructure of the cloud and its tenants, including but not limited to IP addresses, virtual network topologies, and inter-tenant communication patterns [4, 5]. A cloud provider may be reluctant to hand over such data to an auditor, especially third-party auditors who may not be fully trusted [6]. Second, due to the multi-tenancy nature of clouds, auditing results intended for one tenant may inadvertently reveal private information about other tenants. For example, when an auditing report indicates a breach of network isolation, it will typically also include the list of offenders (i.e., reachable VMs belonging to other tenants) as the evidences of such a breach. However, disclosing such information between different tenants may lead to violation of either service level agreements or privacy regulations (e.g. GDPR [7]). Moreover, the leaked information, e.g., another tenant's VM is reachable due to misconfiguration or vulnerabilities, may invite further, more severe attacks. To make our discussions more concrete, we demonstrate such threats, and why existing anonymization techniques are insufficient, through an example.

Motivating Example. Assume a tenant, *Tenant A*, requests the CSP to provide an auditing report on the isolation of its cloud virtual network from other tenants' networks (other security properties and policies will be discussed in Section6.3. To this end, the CSP must provide the auditor with sufficient information about the configuration of virtual networks belonging to the involved tenants in order to verify the reachability between all pairs of VMs.

- As an example of the input data needed for this auditing task, Figure 1.a illustrates an excerpt of the network topology data about *Tenant A* and *Tenant B*. Assume a breach of the network isolation happens since *VM-1* belonging to *Tenant A* is reachable from *VM-2* of *Tenant*



Original Configuration Data Table

Tenant-ID	Net-ID	VM-ID	VM-IP	Router-ID	Next-Hub
TenantA	Net-A1	VM-1	27.0.1.18	RO-1	NH-1
TenantB	Net-B2	VM-2	18.1.66.5	RO-2	NH-1
TenantB	Net-B3	VM-3	18.2.81.2	RO-2	NH-1

(a)

PPE Anonymized Configuration Data Table

Tenant-ID	Net-ID	VM-ID	VM-IP	Router-ID	Next-Hub
TenantC	Net-C1	VM-4	101.13.5.7	RO-3	NH-2
TenantD	Net-D2	VM-5	42.13.6.29	RO-4	NH-2
TenantD	Net-D3	VM-6	42.67.12.1	RO-4	NH-2

(b)

Randomized and Truncated Configuration Data Table

Tenant-ID	Net-ID	VM-ID	VM-IP	Router-ID	Next-Hub
TenantE	Net-G1	VM-7	27.0.X.X	RO-5	NH-3
TenantF	Net-G2	VM-8	18.1.X.X	RO-6	NH-4
TenantG	Net-G3	VM-9	18.2.X.X	RO-7	NH-5

(c)

Figure 1: Excerpt of network topology data: (a) The original data; (b) Anonymized data using equality and PPE; (c) Anonymized data using truncation and randomization.

B due to a vulnerability OSSA-2015-021 [8] in OpenStack (which prevents network security group changes from being effective against running instances). As mentioned above, the challenges are that the CSP is unwilling to share such data in its original form due to concerns that the data may eventually be leaked and misused, and that the CSP must also ensure any auditing result does not disclose information about any other tenants than *Tenant A* itself. Clearly, encrypting the data shown in Figure 1.a using a standard encryption algorithm can address both concerns, but this will also render the data useless.

- Instead, Figure 1.b shows the same network data of Figure 1.a anonymized using property preserving encryption (PPE). Specifically, PPE [9] is applied to IP addresses, and equality preserving encryption [10] is applied to other attributes, such as tenant and VM identifiers (other PPE techniques will be discussed in 3. As a result, network isolation can now be verified using such encrypted data based on the preserved prefix and equality relationships. However, this approach has two main weaknesses. First, most existing PPE techniques suffer from the so-called semantic attacks in which adversaries may extrapolate their prior knowledge about some data to re-identify the encrypted data [11, 12] . Second, more importantly, while the exact values and the format of those attributes are obfuscated, their prefix and equality relationships are preserved to facilitate the auditing task. Unfortunately, the preserved information is also enough for an adversary to completely recover the exact network topology (even though the nodes are obfuscated), as illustrated by the two similar network topologies shown on the right-hand side of Figure 1. Armed with such a complete picture, an adversary could then re-identify the tenants, e.g., based on matching the network sizes to public information about the tenants.
- Finally, Figure 1.c shows the same network data of Figure 1.a anonymized using truncation and randomization. This approach can sufficiently anonymize the data, but the utility is lost. For example, *Net-B2* and *Net-B3* in Figure 1.a are anonymized to be *Net-G2* and *Net-G3* in Figure 1.c. However, their interconnection through the router RO-2 is not preserved in the anonymized data as they are connected each to separate routers *RO-6* and *RO-7* in Figure 1.c. Such a loss of information about the network topology may certainly lead to misleading auditing results about network isolation.
- In summary, Figure 1.b shows that preserving the equality and prefix relationships may facilitate auditing but leak the network topology, whereas Figure 1.c shows that destroying such relationships will protect the data but lose utility. An interesting issue thus arises: *Is it possible to have the best of both worlds (i.e., to protect the network data while still allowing auditing)?*

In this paper, we present, *SegGuard*, a novel anonymization approach that protects the sensitive information in both network data and auditing results, while preserving sufficient data utility to facilitate 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., there is a breach of network isolation) without learning details about other tenants' resources (e.g., the breach is due to reachability from an anonymized tenant's anonymized VMs). Second, we propose a segmentation-based anonymization approach to allow auditing while preventing the leakage of network topology or auditing results. Roughly speaking, our design enables the auditor to generate many segmented views of the network data, and each view contains a few "real" segments (for which the properties required for auditing are preserved) hidden among many "fake" segments (for which the properties are deliberately destroyed); the auditor will perform the auditing task over all the segments, without being able to tell which segments are real, whereas the CSP can secretly extract and integrate the results of those real segments. The following summarizes our main contributions:

- To the best of our knowledge, this is the first work on the unique challenges of protecting both network data and auditing results in a multi-tenant cloud.
- The per-tenant encryption solution ensures no cross-tenant information leakage in the auditing results, which may help ease the privacy concerns of both CSP and tenants in adopting security auditing solutions.
- SegGuard allows semi-trusted auditors to perform auditing without learning detailed configuration or auditing results, which reduces the risk of leaking such sensitive information and enables CSP to outsource the auditing to specialized third-parties.
- We implement *SegGuard* based on OpenStack and evaluate it using both synthetic and real data. The experimental results demonstrate the effectiveness (e.g. less than 1% information leakage for adversaries armed with 50% pre-knowledge, which amounts to a 99% reduction compared to the existing PPE technique) and efficiency (e.g., 62 seconds for anonymizing an infrastructure of 25k VMs).

The remainder of this paper is organized as follows. Section 2 elaborates on the privacy implication of cloud audit data. Section 3 provides background information and the threat model. Section 4 details the proposed solution and Section 5 describes the implementation. Section 6 presents the security analysis and discusses other aspects. Section 7 gives the experimental results. Section 8 reviews the related work and Section 9 concludes the paper.

2 Privacy Implication of Audit Data

In this section, we first discuss the privacy implication of audit data, and then present a concrete case study which shows how sensitive information, e.g., the topology of cloud infrastructures, may be reconstructed from cloud logs.

2.1 Privacy Implications

Today's auditees still choose to trust Third Party Auditors (TPA) or Managed Security Service Provider (MSSP) by providing them the data they need in clear. This is not because they want to, but because they have to, as they don't have any other alternatives to provide them with privacy-preserved data while also keeping the utility high for processing purposes. However, sharing data containing privacy identifiable information in clear have several security and privacy implications (e.g., data confidentiality breaches and privacy regulation violations). In the following, we discuss the privacy implications of audit data with respect to real world incidents, standards and regulations, and the minimal data sharing principle.

Real-world Incidents. Many real world incidents show that privacy breaches may occur while sharing the data with third parties who are usually considered reputable and trustworthy. To name but a few:

1) Employees from reputable TPAs, among the "big 4", in 2016 and 2017, were fired after the leakage of customers sensitive information [13, 14].
2) Employees from a reputable Telecom operator [15], 2013, have stolen and sold customer's data to third parties.
3) A recent survey in the UK shows that 24% of 2,000 employees have leaked confidential business information [16].

In this work, we differentiate between companies like auditing firms (who are interested in protecting their reputation and attract more customers) and individual employees (who may have the financial motivation and the technical means to misuse their privileges and break the confidentiality of the handled customer information), as we will discuss in Section 3.

Data Protection Standards and Regulations. CSPs, as any other company that collect customer data, must follow the regulations and laws about sharing and processing of personal data. These regulations apply not only to the enterprises established in the legislation country but also to enterprises in other countries processing personal data of people inside the legislation country. For example, the GDPR became enforceable beginning 25 May 2018 and states that: *"business processes that handle personal data must be designed and built with consideration of the principles and provide safeguards to protect data and use the highest-possible privacy settings by default"* [7]. Particularly, article 28 prevents any company, including CSPs [17], from sharing customers' data without protection/anonymization when acquiring the services of third parties (e.g., with MSSP). Additionally, the ISO IEC 27002 standard [18] states that If TPAs are involved in an audit, there could be a risk of misuse of audit tools and information being accessed by this third-party organization. Therefore, security controls, such as risks assessment and access restrictions, should be considered. Consequently, services providers choose to completely trust third-parties with their customers' data may risk high fines that can reach 20 million euro or up to 4% of the annual worldwide turnover of the preceding financial year in case of an enterprise, whichever is greater [19]. As a result, anonymizing and protecting the audit data becomes mandatory even with potentially trusted parties.

Minimal Data Sharing Principle (MDS). MDS is a data sharing principle that states that, data must be shared without any extra information exceeding the amount of data required to reach the purpose and conclusion of the sharing process [20]. This principle is used to control the data flow and ensure the unlinkability between different accesses of the data and between the publicly available information. For example, public IP addresses in audit data may potentially be linked to real world user identities, and consequently used to trace back to their activities contained in the same or other related audit data. A possible solution to apply MDS is to anonymize data before sharing it to minimize potentially undesired information disclosure risks.

2.2 Reconstructing Topologies From Cloud Logs

As an example of audit data, logs are valuable assets that need to be protected from destruction, alteration and information disclosure risks. For instance, the MITRE Common Weakness Enumeration (CWE) registry (i.e., CWE-532) states that "information written to log files can be of a sensitive nature and give valuable guidance to an adversary or expose sensitive user information" [21]. In the following, we demonstrate how an attacker can exploit cloud logs in order to gain sensitive information about the cloud virtual infrastructure as depicted in Figure 2.

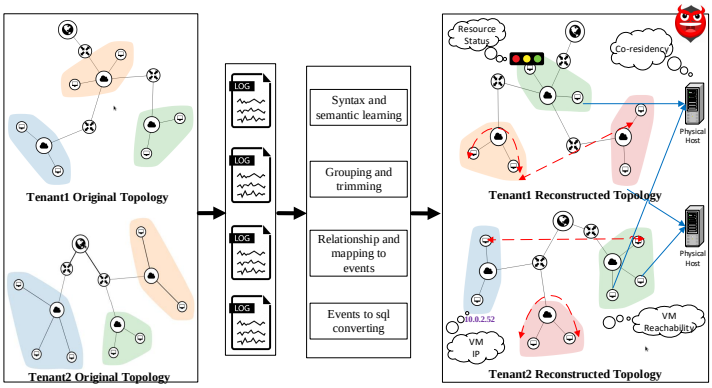


Figure 2: Information inferred from cloud logs.

2.2.1 Background and Goals

OpenStack [22] is an open-source cloud infrastructure management system with a set of software tools for building and managing different services, such as computing (i.e., Nova), networking (i.e., Neutron) and storage resources. OpenStack provides logging services to capture system and user operations within the cloud (e.g., create tenant, router, VM, network, etc.). All log files are stored under the same directory (i.e., /var/log) and each project accesses logs under its own sub-directory (e.g., /var/log/neutron). In the following, we assume an adversary has obtained the row log files generated by the Neutron and Nova services

of an OpenStack cloud with two tenants. We show how the adversary can exploit those logs to extract sensitive information about those tenants in several steps as follows.

- First, the adversary studies the syntax and semantics of the logs, to separate them into different attributes and understand their meanings and relationships.
- Second, the adversary processes the logs by clustering, sorting and filtering their content in order to be able to follow their corresponding real actions on the system with the right order.
- Third, the adversary translates log entries into system events in order to mimic the victim's operations and simulate his/her deployment.
- Finally, the adversary infers sensitive information about the victim's deployment, such as network topology, IP addresses, reachable nodes and co-residency.

2.2.2 Attack Steps

Figure 3 shows an overview of different attack steps which is detailed in the following.

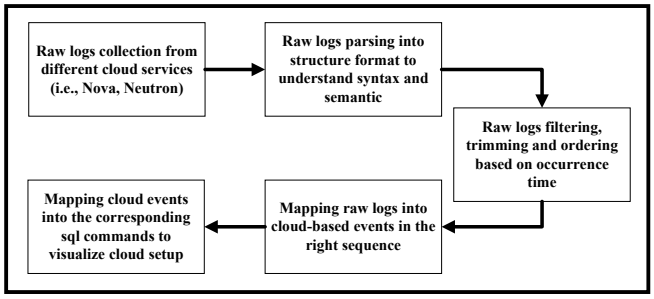


Figure 3: ExLog starting from raw logs and ending with events applied to the adversary's OpenStack databases.

Understanding Log Formats. We first study the general layout of the logs in order to determine the syntax of each attribute and its relative position in order to separate them from each other. He/she will find that the logs contain the attributes shown in Figure 4. After that, the adversary feeds the logs to a log parser to convert them into a structured log file format (i.e., CSV), based on a set of parsing rules. This allows handling the heterogeneity of the logs.

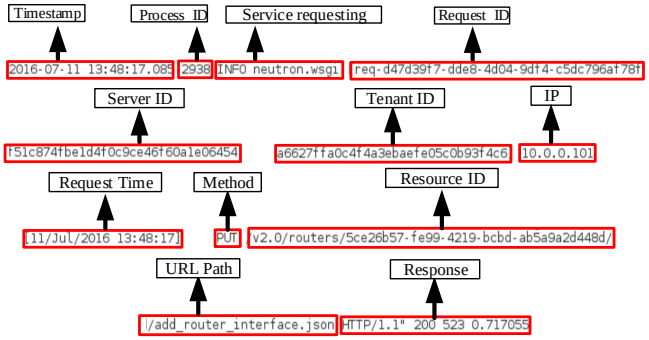


Figure 4: The syntax of OpenStack Nova and Neutron logs

Extracting Log Semantics. After parsing logs into a structured format, the adversary examines each log entry to understand its meaning and how it's correlated to other attributes in the log. He/she realize the following: **i)** events correspond to user API requests, which are associated with the user performing it based on the TenantID ; **ii)** user API requests logged in the system with different method options (i.e., GET, PUT, UPDATE and DELETE). The requests with the *GET* option in the method field are used to fetch information and details about the infrastructure (e.g., the list of VMs), while other options update the infrastructure setup (e.g., delete router); **iii)** the URL PATH field in the logs can identify the type of the event; further details will be discussed in 2.2.2; **iV)** the Host ID field can identify the physical host of the virtual resources, which can be used to determine whether a VM resides on the same physical machine; **V)** there are two timestamp fields in each log entry (i.e., event logging time and request initiation time) that can be used to determine the order of events occurrence; **Vi)** the response field in the logs indicates the success/failure of the event's request (i.e., the *HTTP/1.1" 200*: request is successfully executed).

Grouping and Trimming Log Entries. After understanding the syntax and semantic of the logs, the adversary sorts and arranges log contents to ease the processing as following: **i)** He/she separates the log entries into different profiles based on the TenantID; **ii)** extracts the accessed resource ID (VM IP, Port ID, VM ID, Subnet and Router IDs, etc.), in order to identify which event corresponds to what resources; **iii)** separates the operations into system and user initiated operations (i.e., system initiated operations have special TenantID); **iV)** filters all failed operations (i.e., those which do not have the *HTTP/1.1" 200* response); **V)** sorts the log entries based on the timestamp in order to preserve the right order of operations sequences.

Identifying Cloud Operations. After that, the adversary realizes that all changes and updates that happened in the cloud environment can be recovered from the logs provided if the log entries are correctly mapped into cloud operations. Consequentially, he/she studies and investigates the REST *methods* (e.g., GET, PUT, POST, DELETE) with the specific path information (i.e., called *URL PATH*) to uniquely identify each cloud operation type from each log entry (see Table 1). Nonetheless, there exist some operation types that cannot be uniquely identified by the *method* and the *URL PATH* alone. In order to address this issue, he/she can check the request body of the API request which contains detailed information for each entry.

Method	Resources	Resoruce ID	Action	Oper Type
POST	ports	NaN	NaN	Create port
PUT	ports	port ID	NaN	Update port
DELETE	ports	port ID	NaN	Delete port
PUT	routers	router ID	Add-router-interface	Add interface
PUT	routers	router ID	Rem-router-interface	Remove interface
POST	floating-IPs	NaN	NaN	Create Fl IP
PUT	floating-IPs	VM ID	NaN	Associate FL IP

Table 1: Mapping between log entries and operations' types. Note that, the term 'NaN' is used by OpenStack to indicate the unrepresentable value for the specific fields.

Determining Operations Sequences. The final step done by the the adversary is to generate operations' sequences while

preserving the following properties. First, store different identifiers (e.g., Tenant ID, Resource ID, etc.) and other useful information (e.g., IP address) in each log entry as they will be used to regenerate the same operation in the system. Second, preserve the order of logs entries as they represent the real user and system actions in the cloud (e.g., if the *delete˙router* operation appears before the *create˙router* in the logs, adversary will not be able to determine that the final status of this table is deletion). Finally, he/she preserves the relationship between the operations as they reflect the real status of the cloud.

2.2.3 Re-constructing the victim's Environment

Based on the results the adversary obtained so far, he/she will be able to re-construct the exact topology of the victims' environment. To do so, we assume the adversary has deployed his/her own OpenStack in order to visualize the inferred information. OpenStack has different databases for each service (e.g., Nova, Neutron and each database has different tables corresponding to the different resources in the cloud).

Hence, the adversary can construct the exact setup of any cloud tenant by inserting records into the databases based on captured operations in the correct order of occurrence using SQL commands. For example, the *create˙router* operation is mapped to the following SQL command *"INSERT INTO routers (tenant˙id, id, name, status, admin˙state˙up, gw˙port˙id, enable˙snat, standard˙attr˙id)"*, which will create a router for the corresponding tenant by inserting one record to the table *routers*. Therefore, the adversary creates a script to automatically convert parsed operations into appropriate SQL queries executed in the right order. Figure 2 shows a sample of the inferred information from the logs.

3 PRELIMINARIES

This section provides background information and defines the threat model.

3.1 Background

Property Preserving Encryption (PPE). PPE techniques allow the encrypted data to preserve certain properties, such as equality and prefix relationships, which makes them suitable for our context. Table 2 summarizes some of the existing PPE algorithms. In this paper, we will focus on the prefix-preserving and equality-preserving techniques to protect attributes commonly found in cloud network data, such as internal network identifiers, tenant identifiers, protocols, and IP addresses. However, those PPE techniques only serve as building blocks in our approach, and we can extend it to incorporate other PPE techniques, as will be demonstrated in Section 6.3 through use cases. On the other hand, while PPE is known to have some weaknesses [12], *SegGuard* overcomes them through the segmentation-based anonymization approach, as discussed in more details in Section6 and evaluated through experiments in Section 7.

To facilitate further discussions, we briefly introduce one of the cryptography-based PPE techniques, due to its popularity and simplicity, that is designed to preserve prefix relationships [23]. An anonymization function F is said to be *prefix-preserving*, 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 deterministic (i.e., the same address appearing in different datasets 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 [23]: 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 [24].

Algorithm	Property	Data Type Applicable
Equality-preserving	Equality: same plaintext leads to the same ciphertext	Alphanumeric: ID, password, gender, version
Prefix-preserving	Prefix relationships: preserves IPs hierarchal relationships	IP, age, dates
Format-preserving	Syntax: preserves the syntax of the original data	Timestamp, address, number, protocol, version
Order-preserving	Order: numerical ordering of plaintext is preserved	Timestamp, date, age, other numeric attributes

Table 2: Summary of different PPE algorithms

Compositional Auditing. As mentioned in Section 1, we perform auditing on segments of network data similar to that in [25]. Specifically, instead of auditing the entire input data in one shot, the data is divided into smaller segments and audited separately, then 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. Finally, since our divide-and-conquer approach resembles that found in most existing parallel processing platforms (e.g., MapReduce [26]), an interesting future work is to integrate our approach with such platforms to improve its efficiency and scalability.

3.2 Threat Model

Trust Relationships. *SegGuard* threat model is similar to those used in existing works [27, 28, 29]. First, we define the stakeholders involved in the cloud service model and their trust relationships as follows.

1) The CSP as an organization is interested in protecting its reputation and attract customers. Tenants usually have to trust the CSP for protecting the security related to auditing data and results as part of SLA fulfillment.
2) The tenants are different customers of the CSP, and generally they do not trust each other. They will also be curious to learn about others' resources if allowed.
3) The tenants do not trust the auditors (cloud administrators or third-party), particularly, to have access to their sensitive data (e.g., private IPs, virtual topologies, etc.).
4) The CSP is interested in limiting the amount of information shared with the auditor as long as the auditing tasks can be fufilled due to the serious consequences of potential leakage of such information.

In-scope Threats and Adversarial Model. Similar to existing works (e.g., [30, 31]), we assume the auditor or tenants

will follow given protocols to access anonymized data and perform audit tasks, while s/he is curious enough to infer sensitive information about the cloud infrastructures or the tenants, if the protocols provide him/her such an opportunity. We can distinguish the following types of attacks: **i)** *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 adversary can identify a particular tenant's private resources and information, such as private IPs, open ports, virtual network topology, and inter-tenant relationship. Such an attack could either cause privacy breaches or enable further attacks. **ii)** *Aggregate attacks:* This attack can be done by an auditor. The goal of such attacks is to recover general information about the cloud infrastructure, e.g., the virtual network topologies and communication relationships between different tenants.

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 and publicly accessible nodes); **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).

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 auditing data and results out-of-scope. We focus on information leaked from auditing data and results, and do not consider attacks from other sources, e.g., side channel attacks. Finally, we do not defend against distrusted CSP.

4 SEGGUARD SYSTEM

This section first presents an overview of our approach, and then further details each step.

4.1 Overview

Main Ideas. Before we delve into the details of *SegGuard*, we first build intuitions about its main ideas by considering only three IPs. These are shown in Table 3 (second column under "Original IP"). As discussed in §1, the utility of audit data relies on the preservation of certain properties, including shared prefixes preservation for IP addresses. For the sake of clarity, we will omit for now the discussion about other attributes (we will provide more details later in this section). Below, we present high-level tasks in our approach.

1) First, the CSP PPE encrypts the original IP addresses using a key supplied by each tenant (K_T) and a prefix-preserving encryption algorithm, as shown in the third column of Table 3. 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 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,

	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
Tenant2	51.17.6.5	112.12.42.18	3	201.47.96.23

Table 3: Original data on the CSP side.

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 with three new vectors, V_1, V_2 and V_3 (generated similarly as V_0, and shown in Table 4).

3) Table 4 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, and he obtains different copies of the original data. The resultant IPs are shown in the columns next to their corresponding vectors. As those iterations add up, shared prefixes start to appear. For example, 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) and they had originally shared prefixes. We can observe that, those vectors are chosen in such a way, that exactly one pair of records in every copy is *utility-preserving* (record 1 and record 3 in copy 2, and record 2 and 3 in copy 3) as the number of PPE iterations applied on them (by CSP and then by auditor cumulatively) added up to the same number of iterations. However, although not shown in this toy example, real audit data would also include a larger number of other IPs which also share prefixes in the original IP column, while they do not share prefixes in the copies of the auditor.

4) Next, the auditor performs the auditing task on each copy and obtains the results for all the IPs. From above discussion, we know that, since there is only one pair of utility-preserving records (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 of IPs that also do not share the prefixes in each copy even though they are utility-preserved.

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 them, 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.

	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
Tenant2	201.47.96.23	3	85.50.44.118	1	96.74.34.22	1	33.48.20.25

Table 4: The auditor side (where shaded IPs are utility-preserving and lead to valid auditing results).

To realize the above ideas, the *SegGuard* approach encompasses ten steps as depicated in Figure 5.

Steps1-2: 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 4.2.

Steps 3-4: Next, the CSP aggregates all tenants' data, divides

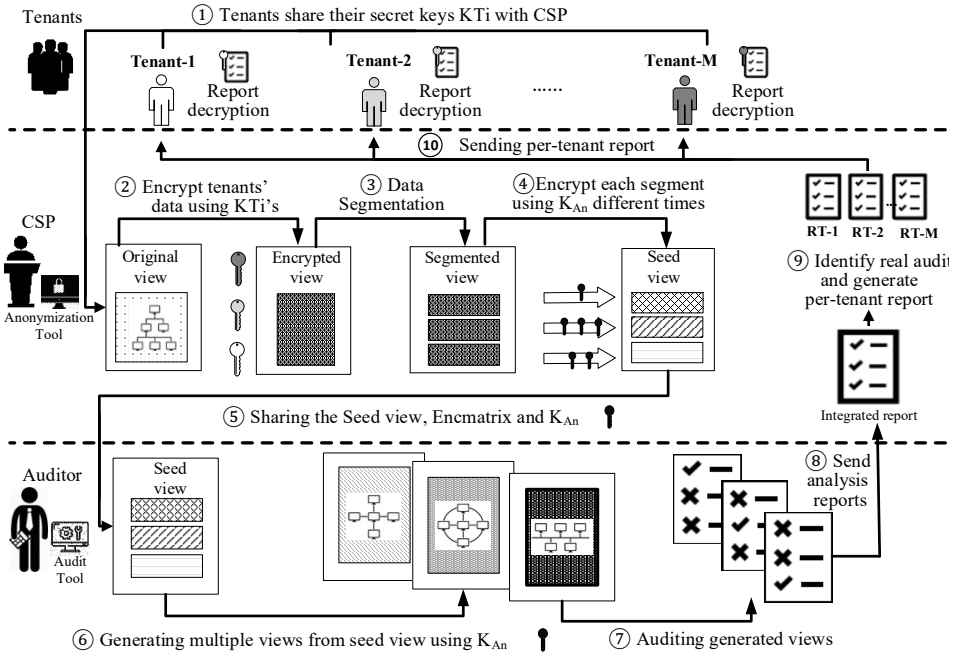


Figure 5: An overview of *SegGuard* approach.

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 view (Step 4), as will be detailed in Section 4.3 and Section 4.4.

Steps 5-6: The CSP shares the generated vectors and seed view with the auditor (Step 5). The auditor then applies those vectors to the seed view in order to generate multiple views (called copies in Table 4) 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 4.5.

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 4.6.

Step 10: Finally, once each tenant receives the audit report, s/he decrypts his/her own data inside the auditing report, as will be detailed in Section 4.6.

4.2 Per-Tenant Encryption Module

Steps 1-2 in Figure 5 provide the first layer of anonymization to protect 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 5), 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 5) 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 5 is an excerpt of the plain network audit data that we will use as a running example throughout this paper. Table 6 illustrates the result of encrypting it using K_{T1} and K_{T2}, respectively. It is observable that the addresses are encrypted while preserving their prefixes, if they originally share the same prefix, and the equality property is preserved for other attributes. An extended version of this example can be found in [32].

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
Ten1	Ne1	VM1	27.0.1.18	1.10.10.1	RO1	RG1
Ten1	Ne2	VM2	27.0.2.9	1.10.10.2	RO2	RG1
Ten2	Ne4	VM4	18.1.5.66	1.10.6.4	RO4	RG2
Ten1	Ne3	VM3	27.0.3.27	1.10.10.1	RO3	RO2
Ten2	Ne5	VM5	18.1.1.43	1.10.6.5	RO5	RG2

Table 5: Original configuration data table.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
9998	X1X	VVV	66.22.10.3	23.15.15.14	WW1	YSE
9998	X3C	MX2	66.22.7.66	23.15.15.16	WW3	YSE
5554	ZQA	SQ1	98.6.36.82	101.2.42.9	WW4	JHQ
9998	HE4	GTQ	66.22.17.34	23.15.15.7	WW2	WW3
5554	CCY	TT2	98.6.41.13	101.2.42.13	WW4	JHQ

Table 6: Encrypted data corresponding to the original configuration table performed by the CSP using K_{T1} (unshaded rows) and K_{T2} (shaded rows).

4.3 Data Segmentation Module

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 ($N_{seg-view}$), as chosen by the CSP. The CSP then performs the following operations:

1) **Computing number of views:** The CSP computes the number of views N_{views}, that he need to communicate to the auditor based on the following formula:

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

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 6), then, based on Equation 1, the auditor has to generate 10 views. The details about the selection of these parameters and their impact are studied Section 6.3, Section 7.4.

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 [33], (Step 3 in Figure 5) 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 to keep track of tenant's assets. The output of this step is the segmented view data, denoted by $SegNet$.

Example 2. Table 7 shows the result of sorting and segmenting the data, $SegNet$, where each record in Table 6 represents one segment.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
9998	S3A	VVV	66.22.10.3	23.15.15.14	WW1	YSE
5554	ZQA	SQ1	98.6.36.82	101.2.42.9	WW4	JHQ
9998	X3C	MX2	66.22.7.66	23.15.15.16	WW3	YSE
5554	CCY	TT2	98.6.41.13	101.2.42.13	WW4	JHQ
9998	HE4	GTQ	66.22.17.34	23.15.15.7	WW2	EQW

Table 7: $SegNet$ corresponding to the encrypted and segmented data of Table 6 performed at the CSP side.

4.4 Utility Preserving Anonymization Module

The CSP first generates a set of utility parameters, (i.e., V_0 - V_3 in the example discussed in Section 4.1), to be used to ensure the utility of the seed view 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 $SegNet$ to obtain the seed view before transferring it to the auditor (Step 4 in Figure 5). 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 $EncMatrix_p$ (resulting of a permutation applied on EncMatrix as discussed later on) 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:

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**
    $random[j] =$ **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

$$EncMatrix[i][j] = VP[i][j] - V_{Random}[j] \quad (2)$$

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 Section 6). The following example shows how the parameters are generated.

Example 3. For a data with 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): $V_{Random} = [3; 5; 2; 7; 4]$
2) A set of ten vectors $\{VP_i\}_{i<=N_{views}}$ (kept secret by CSP):

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

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

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 elements 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 SegNet. The i^{th} segment is then encrypted $V_{Random}[i]$ times using K_{An}. After that, the rows of $EncMatrix$ and the segments of the anonymized $SegNet$ are permuted together randomly to hide the position of the real segments in the generated views. The result of this

Algorithm 2 UtilityPreservAnonym

V_{Random}=**GenerateVR ()**
VP=**GenerateVP ()**
K_{An}=**GenerateKey()**
for $(int\ i = 1; i <= N_{views}; i++)$ **do**
  for $(int\ j = 1; j <= N_{seg}; j++)$ **do**
$EncMatrix[i][j] = VP_i[j] - V_{Random}[j]$
for $(int\ i = 1; i <= N_{Seg}; i++)$ **do**
  for $(int\ j = 0; j <= V_{Random}[i]; j++)$ **do**
    $SegNet_e$=**EncryptSegNet(**$SegNet, K_{An}$**)**
$EncMatrix_p, SegNet_p$=**Permute**$(EncMatrix, SegNet_e)$ ▷ permutation function
return $VP_i, K_{An}, EncMatrix_p, SegNet_p$

stage is denoted as $SegNet_p$. Algorithm 2 presents high-level details of this step.

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 $SegNet_p$ 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	4.32.74.8	WDS	YSQ
8745	12W	W5F	34.16.5.40	13.20.15.14	W5R	RDP
5684	OI1	WE2	75.81.15.34	55.1.17.22	LA4	MVZ
6571	Q4E	WQ1	74.99.41.52	86.54.1.32	W3S	UZQ
9865	IO1	KK2	162.2.10.12	99.65.8.1	QW1	JAQ

Table 8: $SegNet_p$ after permutation and encryption.

4.5 Multi-View Generation Module

At this stage, the auditor receives $EncMatrix_p$, $SegNet_p$, and K_{An} from the CSP and generates the multiple views as follows. Based on the number of columns in $EncMatrix_p$, $SegNet_p$ 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 $EncMatrix_p$) is 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 indicated in the corresponding value (i, j) in the $EncMatrix_p$ (Step 6 in Figure 5).

Example 5. Table 9 and Table 10 respectively show the first and second views generated by the auditor. The shaded segments are those parts that are real, the equality property and shared prefixes are preserved and their auditing would lead to valid results. The unshaded segments, indistinguishable from the real ones, would lead to invalid audit results. Note that those invalid results would not impact the validity of the results from the real segments and will be safely discarded by the CSP as it will be shown in Section 4.6.

Once all views are generated, the auditor analyzes them (Step 7 in Figure 5), and sends the reports for each view back to the CSP (Step 8 in Figure 5), where each report is identified using the column index in $EncMatrix$ used to obtain the corresponding view.

Tenant-ID	Net-ID	VM-ID	VM-Pri-IP	VM-Pub-IP	Router-ID	Next-Hop
2478	11E	I3I	63.89.65.15	48.45.19.1	KL6	MZN
5678	4F3	9KL	89.98.45.6	82.112.10.5	OP3	ASP
7231	LX2	WSA	87.65.7.2	33.10.22.7	WDS	MNO
6761	CNN	M8P	36.74.51.46	91.79.63.3	HN3	L4U
7231	NJ2	KFD	87.65.7.65	33.10.22.9	NW1	GH3

Table 9: 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	111.80.7.19	PLM	GH9
7238	1W2	VCS	14.15.89.54	241.1.60.7	PP9	3KY
9651	DDF	MNI	25.6.99.11	92.99.10.9	HMH	IS1
3456	4TF	WSA	18.2.6.12	146.5.65.14	NF5	T5R
7676	GGF	WQ1	17.2.9.24	111.80.7.10	NJ1	MQF

Table 10: Second view generated by the auditor. Shaded segments are real segments and unshaded rows are fake ones.

Example 6. Figure 6 shows the constructed virtual topologies from the first two generated views in Tables 9 and 10. As we can see, the generated topologies are different from the original topology 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 [34]). 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).

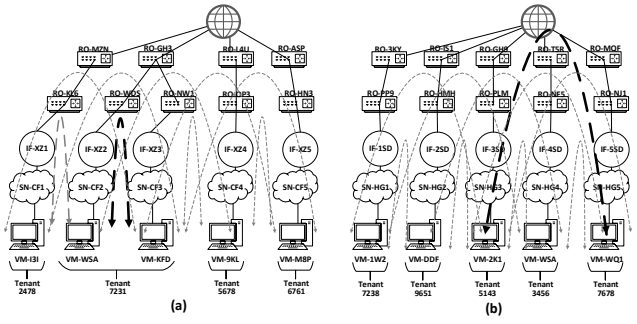


Figure 6: Network topologies inferred from two generated views (together with the encrypted SGR table not shown here): (a) topology related to first view in Table 9, (b) topology related to second view in Table 10. The gray dashed lines are fake and boldface dashed lines are real.

4.6 Per-Tenant Report Integration Module

This module runs at the CSP side and takes as input the reports received from the auditor and their identifiers as described in the previous section, recovers the correct results from all reports for all tenants, and then prepares a per-tenant report (Step 9 in Figure 5). 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 $SegNet$ records are used to identify the real segments in each view. Thus, view (report) IDs, tuples of information from $SegNet$, 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.

Example 7. Table 11 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 11: 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 the 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 11.

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 protecting the sensitive attributes of other tenants. Finally, each tenant T_i decrypts the report using his/her key KT_i (Step 10 in Figure 5).

Example 8. Table 12 shows the final auditing reports forwarded to Tenant1 (*Tenant-ID: 1234*). Table 13 shows the final decrypted auditing reports of Tenant1 (*Tenant-ID: 1234*) using K_{T1}. Based on Table 13, 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 12: 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	Reachable
1234	CD3	27.0.3.27	1234	CD2	27.0.2.9	Reachable

Table 13: Final decrypted report for Tenant1. Shaded cells were decrypted by Tenant1.

5 Integration into OpenStack

In this section, we detail the implementation of *SegGuard* and its integration into OpenStack.

Background. OpenStack [22] 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 [35]. 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.

SegGuard Integration into OpenStack. Figure 7 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 Nova and Neutron configuration data, and with the CSP to obtain the utility parameters required by SegGuard as explained in Section 4.4. 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:** Once the data collected filtered and stored in *SegGuard* database, this engine performs *SegGuard* operations at the CSP side, where it takes as input the encryption keys of the tenants, the key shared with the auditor, the number of segments and real segments per view, then segments the data accordingly. It also generates the different utility parameters, which are the secret random vector V_{Random}, the secret vectors VP_i (stored locally), and the encryption matrix $EncMatrix_p$ to be shared with the auditor. The segmented data is finally encrypted where each

segment is encrypted by the CSP using the auditor's key with different number of iterations as explained in Section 4. The data calls and storing is done through SQL and bash scripts, while the encryption algorithms are implemented in C++. The output of this module is the seed view and $EncMatrix_p$, also stored in the database.

3) **Anonymization Evaluator Engine:** This component performs two main steps. First, it plays the role of data auditor and takes as input the seed view 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 pushing the generated views databases into the OpenStack databases in order to visualize the resulted topology through the web based user interface service, namely, Horizon [36].

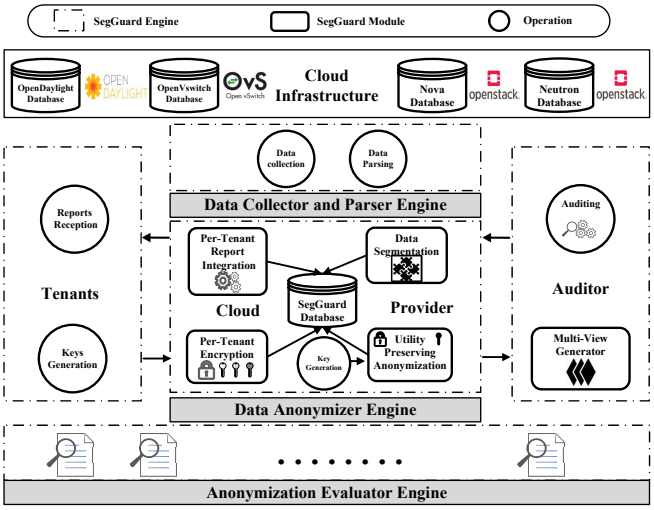


Figure 7: A high-level architecture of *SegGuard*.

Figure 8 shows four screenshots from the OpenStack Horizon web interface illustrating the network topologies generated at different stages of the anonymization process. In our system, these are the outputs of the *Anonymization Evaluator Engine* when it uploads anonymized data into the OpenStack database service to enable this visualization. Specifically, Figure 8.(a) corresponds to the original topology while Figure 8.(b) depicts the topology inferred from the seed view generated by $SegGuard$. One can easily observe that the seed view does not preserve any topological information of the original data. Figure 8.(c) and Figure 8.(d) corresponds to the network topology derived from two randomly selected views (the third and the fifth one, respectively) generated at the auditor side. The encircled portions in the topology of these two last views correspond to the real segments (which cannot be distinguished by the auditor).

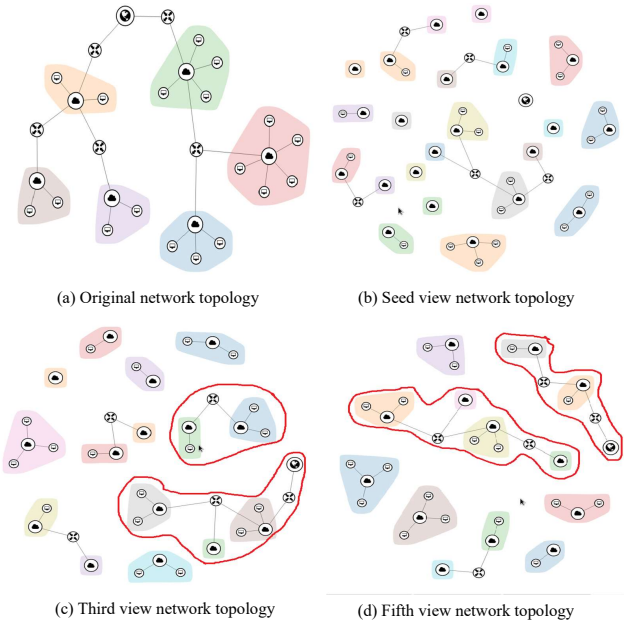


Figure 8: Four network topologies displayed inside the Openstack Horizon dashboard corresponding to (a) the original topology, (b) the seed view, (c) the third view and, (d) the fifth view generated by the auditor.

6 Analysis and Discussions

In the following, we first analyze the security and the utility of our solution and then discuss other aspects.

6.1 Security Analysis

In the following, we analyze the security offered by SegGuard w.r.t. the sensitive attributes, the network topology, and the auditing results. The security of *SegGuard* relies on the resistance of both the seed view and those views generated by the auditor against various attacks. Those attacks may either aim to re-identify encrypted data (semantic attacks) or learn the relationships between data records (network topology inference). Note that an adversary (at the auditor side) can only identify valid auditing results after one of those attacks is successful.

Resistance Against Network Topology Inference Attacks. As explained in Section 4.1, the seed view generated by the CSP does not preserve the equality property or the shared prefixes, which makes it resistant to the topology inference attack, and auditing the seed view will not lead to any useful result, either. In addition, none of the generated view would preserve those properties, except for a few records in the real segments (which the adversary cannot identify). Finally, the properties are not preserved across any real segments inside different views. Therefore, the adversary has a slim chance to recover any useful information from those views, as will be shown through experiments in Section 7. The following first provides a detailed analysis of the brute force attack that guesses the critical utility parameters to recover real segments. To this end, the adversary has two choices: Either **i)** guessing V_{Random} and then using Equation 2 with $EncMatrix$ to compute all VP_i, or **ii)** guessing all VP_i directly, 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:

$$P\left(V_{Random}\right) = \frac{1}{D} * \frac{1}{(D-1)} * \ldots * \frac{1}{D - N_{Seg}} \quad (3)$$

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:

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

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 (e.g., 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 the domain of size D, so the probability to generate any value in the matrix would be $\frac{1}{D}$, which is not considered as information leakage according to the theory of secure multiparty computation [37] since it can be simulated in a polynomial time.

Resistance Against Semantics Attacks The property preserving encryption techniques are well known to be vulnerable to the so-called semantic attacks in which an adversary may extrapolate pre-knowledge about some deliberately injected data to reidentify other encrypted data [20]. However, such an attack is less likely to be achieved in our context, since injecting network configuration data cannot be easily performed across tenants. Moreover, for frequency analysis attacks [12] in which the attacker performs matching between a probabilistic model of prefixes constructed based on pre-knowledge, and the anonymized IPs, respectively. Our segmentation-based anonymization approach makes it significantly more difficult to construct the probabilistic model based on the anonymized data. Nonetheless, since this is one of the most powerful semantic attacks, we have implemented it to further evaluate its impact on SegGuard in Section 7. For this purpose, we define the *Information Leakage* resulting from this attack as the number of deanonymized IP addresses after applying the frequency analysis attack on the anonymized view generated by the *SegGuard* approach over the total number of the original IP addresses.

6.2 Utility Analysis

First, the use of deterministic and property-preserving encryption ensures the consistency and validity of auditing results. Second, the use of per-tenant encryption using tenant-specific keys has an added benefit, i.e., it guarantees that attributes with the same value (such as private IPs) from different tenants are mapped to different values to prevent loss of information during data aggregation. Third, the segmentation-based anonymization approach guarantees that the auditing results are equivalent, from both completeness and correctness point of views, to the results of auditing the originally anonymized data (which is equivalent to the union of real segments in all the views), as long as the auditing task is compositional. In summary, the data utility is sufficiently preserved under *SegGuard* for all in-scope auditing tasks.

6.3 Discussions

Security Policies. In this paper, we have focused on network isolation and reachability to show the applicability of our approach. SegGuard can be extended to audit other security policies as follows. First, since security policies (e.g., network isolation) are mostly composed of lower level properties (e.g., reachability), a policy can be supported if all the involved properties are also supported. Second, SegGuard can support a security property if it is compositional (see Section 3) and there exist its corresponding PPE techniques for preserving its required properties (e.g., equality relationships and shared prefixes). Table 14 shows examples of supported security policies and related security properties.

Cloud level	Security policy	Security property
Infrastructure	Resource isolation	VM co-residencey, port consistency [38]
Network	Network isolation	loops, black-holes, reachability [39]
Application	User privilege	functional behaviour and security-related [40]
User	Unauthorized access	common ownership, minimum exposure [41]

Table 14: Compositional security properties

Computational Cost and Benign Auditor. By preserving data utility, our solution essentially imposes a tradeoff between security and computational cost. However, in practice, the CSP can adjust these based on the security and privacy requirements and the amount of computation cost s/he can afford. Specifically, the parameters N_{seg} and $N_{seg-view}$ (Section 4.3) determine the number of views to be generated by the auditor, which is related to the security level and the computational cost. If an auditor is more trusted, e.g., a cloud administrator, the CSP can reduce the number of views to be generated by either decreasing the number of utility segments per view or choose a small number of segments in the first place. This tradeoff is studied in Section 7.

Colluding Adversaries. First, if two tenants choose the same key accidentally or intentionally, the CSP could detect and revoke those keys or choose different keys for them. Second, if a tenant and an auditor collude, it would be equivalent to an auditor having more pre-knowledge (i.e., knowledge about all the data of the colluding tenant). As our experiments will show, *SegGuard* is robust against such adversaries.

Auditing Static Cloud Configurations. *SegGuard* supports auditing on cloud configurations collected as snapshots through our data collection engine (as described in Section 5). Our experimental results (in Section 7) show that our approach is practical for periodically anonymizing relatively large clouds (e.g., about one minute for 26 thousand VMs). The incremental option for *SegGuard* is an interesting future work which can allow anonymizing a small amount of configuration changes more efficiently.

Communication and Storage Costs. In the third-party auditing case, transmitting the anonymized audit data and other necessary parameters may involve certain communication cost. However, our design of the seed view means only one copy of the audit data will need to be sent over the network, even though the auditor may later generate a large number of views on his/her site. On the other hand, if the CSP chooses to perform in cloud auditing, then no data would need to be sent over the network. In both cases,

the storage cost for the multiple views could be a concern especially when the number of views is large. One way to mitigate this is to generate, audit, and then delete each view before generating the next, assuming the auditing results have lower storage requirements.

Untrusted CSP. We note that an alternative approach here is to move the auditing result integration step from the CSP to the auditor side. This can potentially reduce the amount of trust needed in our approach and enhance the effectiveness of SegGuard. This is because the CSP has a higher interest in fixing and hiding the existing system bugs and misconfiguration instead of sending it back to the tenant as this might affect his reputations. We consider the possibility and it's integration as a future work.

6.4 Use-cases

We discuss how *SegGuard* may apply to other use cases.

Network Trace Anonymization. Network traces analysis plays an important role in various network security tasks including cyber attacks investigation and vulnerabilities assessments. Generally, network traces include packet headers and related information such as timestamps. *SegGuard* can be easily adapted to anonymize network traces. For instance, in addition to equality and prefix-preserving encryption, order-preserving encryption can be used to anonymize timestamps and packet sizes consistently. Since network traces do not have the tenant concept, the per-tenant encryption module should perform encryption based on the origin of the traces, e.g., same hosts or subnets are encrypted under the same key. Also, the data segmentation module needs to ensure the network traces from the same origin are distributed among different segments. In Section 7, we will apply *SegGuard* to real world network traces in our experiments.

Cross-Domain Firewalls. Privacy issues can become a real concern for firewalls separating different administrative domains. This stems from the fact that firewalls policies and rules may disclose confidential information such potential security breaches, holes and misconfigurations. As firewall rules use the same attributes as in network traces, *SegGuard* can be adapted to cross-domain firewall rules by adding support for encrypting port number data attribute field and segmenting rules based on the subnets to which the firewall rules are applied.

Other Applications. *SegGuard* may be extended to Central Authentication Services (CAS) applications which consist of three main components, client web browser, web application requesting authentication, and CAS server. *SegGuard* can be used as a CAS service after some modification where the client web browser can play the role of the Tenants, the web application as the CSP, and the CAS server as the auditor. Also, *SegGuard* can be used to preserve the privacy of the actual physical setup of data centers and data farms as we will demonstrate in Section 7.5.

7 Experiments

This section presents experimental results measuring the effectiveness and efficiency of $SegGuard$.

7.1 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. Furthermore, we use a synthetic dataset collected from our testbed deployment of OpenStack (version Kilo) cloud environment in order to simulate a large cloud deployment of 25.2K different VMs.[1] Details of those datasets are provided in Table 15. Finally, all our experiments are performed using a PC with Linux 13.6 64-bits, Intel Core i7 CPU, and 16GB memory.

Dataset	Statistics			Source	Type/Use
Dataset-1	Number of Distinct IPs			Real	Network trace data/ Validity against semantic attacks
	10K				
Dataset-2	Physical machines	Tenants	VMs	Real	Virtual network configuration/ Studying the applicability of SegGuard
	22	37	4377		
Dataset-3	Virtual routers	Subnets	VMs	synthatic	Cloud configuration data/ Evaluate efficiency
	1200	3200	25200		

Table 15: Details of the datasets used for experiments

7.2 Information Leakage under Semantic Attacks

In this section, we evaluate the effectiveness of $SegGuard$ by examining the *information leakage* (Section 6.1) of our solution under the *Frequency Analysis Attack* defined in Section 6.1, based on the data generated by both the CSP (seed view) 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. The results are shown in Figure 9 and detailed below.

Impact of Varying the Number of Segments. In this experiment, we analyze the information leakage for a 10K VMs data set against an adversary with 25% knowledge of the original data while varying the number of segments. Figure 9.(a) 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 view based on his/her pre-knowledge, s/he is not able to find them on the anonymized version of the view.

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 9.(b) shows that the information leakage increases steadily when the number of real (utility-preserving) segments increases per view. This is expected because, having a large number of utility preserving segments per view increases the chances that the IPs that were originally sharing the same prefixes are also sharing again the same prefixes in the same view. This makes it easier for the adversary to deanonymize them. In the worst case, when we have 10 real segments out of 10 segments, the entire view is prefix-preserved (and the equality property holds) and thus an adversary can

1. This is considered a large cloud according to [42] as 94% of OpenStack deployments have less than 10K distinct IP addresses.

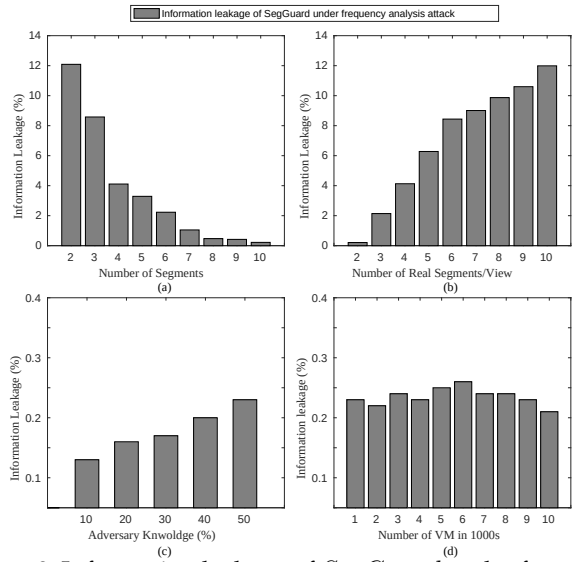


Figure 9: Information leakage of $SegGuard$ under frequency analysis attack while varying (a) the number of segments, (b) the number of real segments per view (c) the adversary knowledge, and (d) the number of VMs in the cloud. In all cases, except the varying parameter, we fix the other parameters as follows: the adversary knowledge to 25%, the number of VMs to 10K, the number of segments to 10, and the number of real segments per view to 2

deanonymize 1.2K of the IPs, which represents 12% of the entire view. However, when the number of real segments is only two, the adversary can only deanonymize 22 IPs from the entire view, 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 Section 6.3, for a given number of utility-preserved segments per view, the larger number of segments, the lower the level of information leakage is and thus the higher would be the auditing cost (as the number of views to be audited increases).

Impact of Varying the Adversary Knowledge. In this experiment, we analyze the information leakage when we have 10 segments and two real segments per view, while varying the adversary knowledge from 10% to 50% of the 10K VMs. Figure 9.(c) 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 cause the same IP sharing the same prefixes to be encrypted with different number of layered encryption. Therefore, when an adversary builds the probabilistic model for both original and anonymized views, s/he cannot find matches across segments. Finally, for the retrieved deanonymized IPs, there could be distinct IPs that are within the adversary knowledge and appear once in the anonymized view.

Impact of Varying the Number of VMs. Figure 9.(d) shows the information leakage as a function of the number of VMs in the view for a fixed adversary knowledge of 25%, 10 segments and two real segments/view. The figure shows that the percentage of information leakage is independent of the size of the view. 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 view size of 6K VMs. Thus, there is no significant impact of different sizes of the view on the level of security and privacy offered by our approach.

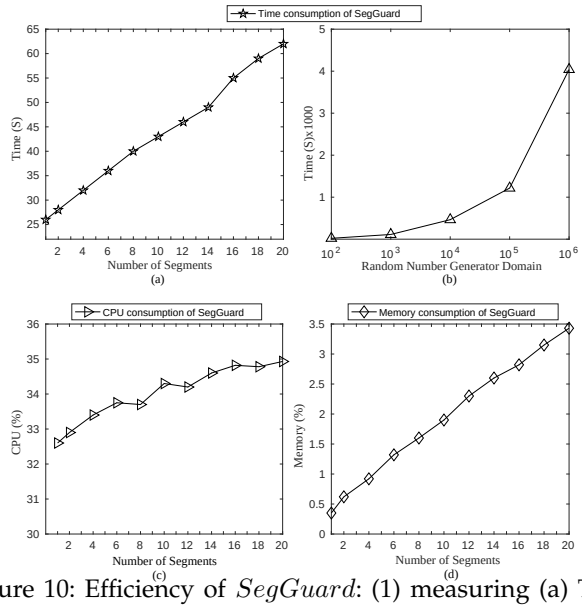


Figure 10: Efficiency of $SegGuard$: (1) measuring (a) Time (c) CPU consumption (d) and Memory consumption while varying the number of segments and (2) measuring (b) Time while varying the size of the domain of the random number

7.3 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 measure the time as well as memory and CPU consumptions while varying the number of segments and the range of random values, respectively. Figure 10 shows the obtained results. Figure 10.(a) shows that the time required to prepare the seed view linearly increases with the number of segments. The time difference between having one segment and 20 segments is about 36 seconds. Figure 10.(b) 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 sufficient even for a reasonably chosen random domain size. For instance, the required time to generate the seed view 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 view. 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.

Figure 10.(c) 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 10.(d) 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. The results show that $SegGuard$ does not require excessive resources and can perform anonymization under a minute for large datasets.

7.4 Privacy and Computation Cost Trade-off

As our solution gains privacy and utility at the cost of more computations, we evaluate the trade-off between privacy and computational overhead in terms of time, CPU and memory consumption, while varying the number of views. We also compare the information leakage between our solution and the existing PPE solution [23] while varying the adversary knowledge in order to demonstrate how $SegGuard$ can overcome the limitations of those solutions. Figure 11.(a) shows that the time required to generate all the views increases while the information leakage decreases. There is a sharp decrease in information leakage before about six views, and the difference in time between having one view and 45 views is about 23 minutes. Figure 11.(b) shows the CPU usage required to generate all the views increases while the information leakage decreases. The CPU difference between having one view and 45 views is about 7.2% which is not significant. Figure 11.(c) shows a similar trend for the memory consumption. The memory difference between having one view and 45 views is about 7.6% which is not significant either. Figure 11.(d) compares $SegGuard$ to the PPE solution preseneted in [23] which shows that an adversary armed with 50% of pre-knowledge of the original data can leak about 22% of the PPE-anonymized data. However, s/he can only leak 0.35% under our solution when the number of segments selected is 5 and the number of real segments/view is 2.

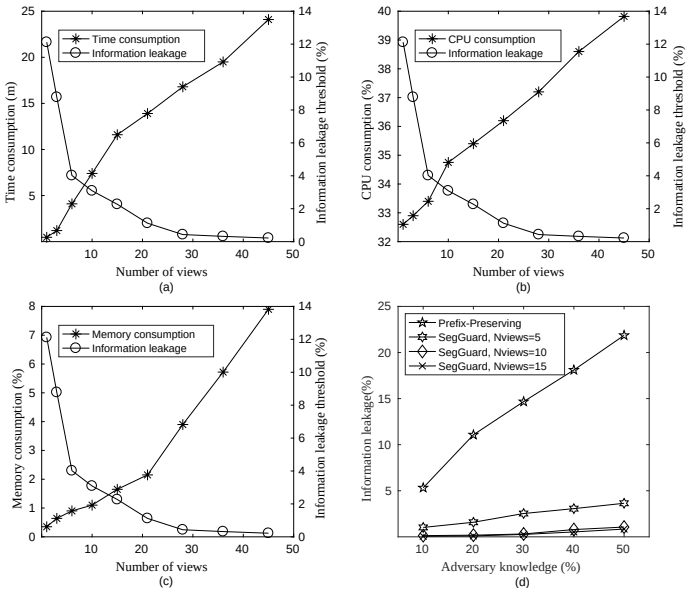


Figure 11: Privacy and Computation cost trade-off of $SegGuard$ under frequency analysis attack while measuring (a) Time, (b) Memory, and (c) the CPU consumption and varying N_{views}. Comparing (d) $SegGuard$ to PPE anonymization. In all cases, except the varying parameter, we fix the other parameters as follows: the adversary knowledge to 25%, the number of VMs to 10K, the number of segments to 10, and the number of real segments/view to 2

7.5 Topology Preservation in a Real Cloud

For this experiment, we apply $SegGuard$ to the data collected from a real world community cloud to examine the impact of anonymization on the network topology from the auditor (adversary)'s point of view. Figure 12.(A) illustrates the actual topology of this cloud (many details are omitted due to space limitations), where it is composed of two racks, $Rack_1$ and $Rack_2$, connected to two edge switches, namely $Edg_1 1$ and $Edg_1 2$, which are connected to two aggregate switches, namely *Agg11* and *Agg12*. Each rack consists of 11 physical servers and hosts assets from totally 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 have randomly selected two views generated by the auditor and visualized their corresponding topology as illustrated in Figure 12.(B) and Figure 12.(C). As we can see, the two anonymized topologies are significdantly different from the original one, e.g., the total number of tenants in the former figure is 131 while in latter it is 107, both of which are significantly different from the real settings. This is due to the fact that the tenants' identifiers are spread over the four segments and are 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, and therefore it will appear as a different tenant in each view. Similar changes also happen to the identifiers of the virtual resources, the physical servers, the racks, as well as the edge and aggregation switches. In summary, the auditor (adversary) will not be able to extract much useful information from these anonymized topologies.

7.6 Complexity Analysis of SegGuard

In this section, we analyze the computation and communication overhead resulted from $SegGuard$ on both the CSP side and auditor side. Whereas, $P(n)$ and $E(n)$ refers to the computation cost resulted from the use of prefix-preserving encryption and the AES encryption algorithms, respectively. Also, $A(n)$ and $G(n)$ represent the auditing task and per-tenant reports generation computation cost, respectively. Also, T is the total number of tenants that we have in the anonymized data. In table 16 we summarize the overhead resulted from all building blocks of $SegGuard$.

8 Related Work

The widely used techniques for anonymizing network data, include truncation, randomization, quantization, and pseudonymization, has been presented and investigated in several recent works as they are supporting a diverse

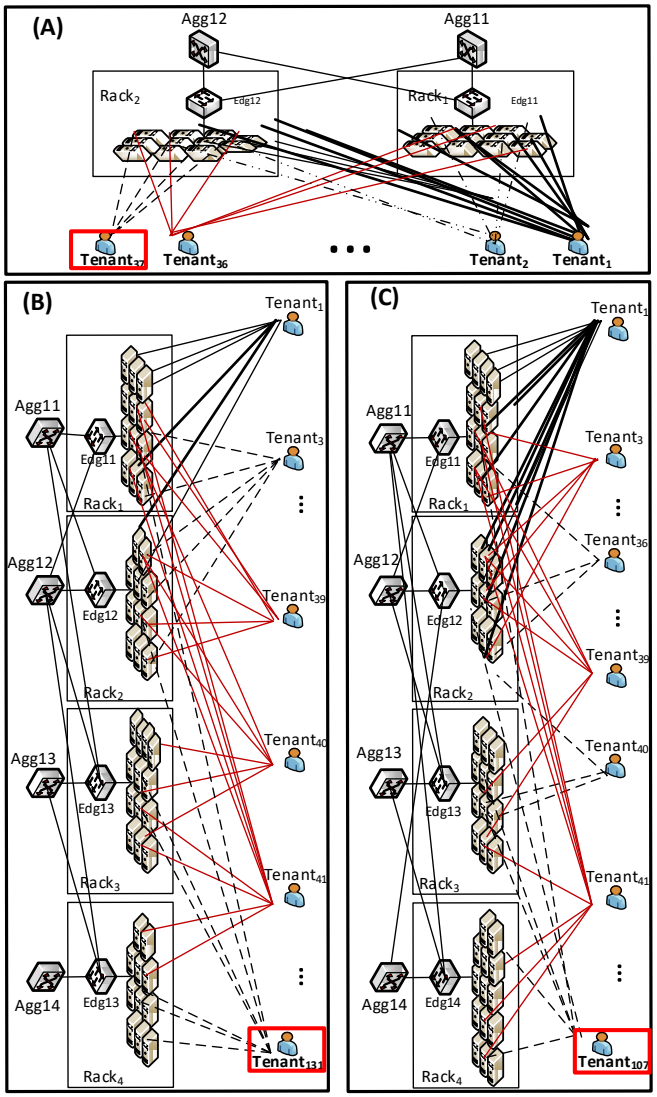


Figure 12: Excerpts of the topology of a real cloud data center (A) as built from the original data set. Figures (B) and (C) show the topology corresponds to two different views selected randomly from the views generated by $SegGuard$. The red rectangle indicates the total number of tenants.

Actor	Operation	Computation Overhead	Communication Overhead
CSP	Initial anonymization	$P(n) + E(n)$	–
	Round-robin scheduling	$O(1)$	–
	Data segmentation	–	–
	Vectors generation	$(2 \times N_{views} + 1) \times O(n)$	$N_{views} \times O(n)$
	Seed-view generation	$\frac{\sum_{n=1}^{N_{Seg}} V_{Random}}{N_{Seg}} \times (P(N) + A(n))$	$O(n)$
	Reports retrieving	–	$O(n)$
	Per-Tenant reports	$T \times G(n)$	$O(n)$
Auditor	Multi-views generation	$\frac{\sum_{i=1}^{N_{Seg}} \sum_{j=1}^{N_{Seg}} EncMatrix[i][j]}{N_{Seg}} \times (P(N) + A(n))$	–
	Auditing	$N_{views} \times A(n)$	–
Tenant	Key sharing	–	$O(n)$
	Report Decryption	$E(n)$	–

Table 16: Summary of $SegGuard$ overhead

set of features suitable for applications such as encrypted search, ML and clustering on encrypted data [43, 44, 45]. Truncation and randomization [46] effectively destroy the semantics of the field they are applied to. Quantization techniques [47], such as limiting the precision of timestamps, are applied to reduce the information gained about the identity of the workstations from timing attacks [48]. One of the most widely used techniques, pseudonymization [49], replaces IPs found in the data with linkable, prefix-preserving pseudonyms. These pseudonyms preserve the hierarchical relationships found in the prefixes of the original addresses. The underlying goal is to enable the analysis of packets generated by hosts, or entire prefixes, without providing the actual IPs. Those existing techniques are often vulnerable to the so-called semantic attacks [11, 12, 50]. Specifically, Naveed et al. [11] studies the security of data encrypted using such PPE techniques, and presents different types of attacks that may allow an adversary to decrypt a large portion of the data based on his/her pre-knowledge. Similar attack are also discussed in [12] where it is shown that an adversary with a pre-knowledge can deanonymize selected addresses or subnets. The (k,j)-obfuscation technique introduced in [42] addresses 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. [51] 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. [52] provides 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. The differentially private analysis method first adds noises to analysis results and then publishes such aggregated results [53]. Although this method may provide privacy guarantee regardless of adversarial knowledge, the perturbation and aggregation prevent its application to our context since auditing usually demands detailed data records rather than statistics. In summary, directly applying those existing techniques to our context may either sacrifice data utility and prevent security auditing, or cause security issues, such as leaking network topology, leaking auditing results, or being vulnerable to semantic attacks.

For security auditing in clouds, there exist many solutions for auditing the virtualization infrastructures of clouds in terms of structural properties, such as CloudRadar [54]), NoD [55], and TenantGuard [56], which can all check all-pair reachability in networks for large cloud data centers. Congress [22] is an open project for OpenStack platforms which enforces policies expressed by tenants and then audits the state of the cloud to check its compliance. Security auditing is also studied for general networks, e.g., VeriFlow [57], NetPlumber [38], and AP verifier [58] can all perform near real-time auditing of virtual networks. However, to the best of our knowledge, those existing works have largely either ignored the issue of anonymizing the input data and auditing results or considered it a future work.

9 Conclusion

We presented a novel anonymization approach that limited the leakage of sensitive information from either the auditing data or auditing results in a multi-tenancy cloud, while preserving sufficient utility for effective security auditing. Our solution could allow CSP and tenants to adopt security auditing solutions with more confidence and to reduce the amount of trust the CSP must place in third party auditors.

Limitations. The main limitations of our work and corresponding future directions are as follows. First, the

	Input Type						Anonymized Fields					Limitations			
Technique	**Tcpdump**	**Netflow**	**NFdump**	**pcap**	**Life inter**	**TSH**	**NF fields**	**IP**	**Port**	**Header**	**Payload**	**Semantic attack**	**Leaking topology**	**Leaking results**	**Utility loss**
Pref-Pres [11]	✓	✓			✓		✓	✓				✓	✓	✓	
Permutation [46]		✓	✓	✓		✓	✓	✓	✓						✓
Truncation [46]	✓	✓	✓	✓		✓	✓	✓	✓						✓
Hashing [11]	✓	✓			✓		✓	✓						✓	✓
Hiding [12]	✓	✓			✓		✓	✓				✓	✓		
Randomization [46]	✓	✓		✓		✓		✓		✓					✓
Pseudonyization [49]		✓	✓		✓			✓				✓	✓	✓	
Quantization [47]	✓	✓			✓			✓		✓	✓				✓
Shifitng [12]	✓	✓	✓				✓	✓	✓						✓

Table 17: Comparing exisiting network data anonymization techniques when applied to our context (the symbol ✓ indicates an applicable feature or limitation).

integrity of auditing data and results are not addressed in this paper and could potentially be addressed by adapting existing integrity mechanisms; we also fully rely on the CSP and assume an honest but curious adversary model, and weakening such assumptions by applying trust computing solutions is an interesting future direction. Second, we have mainly focused on equality and prefix preserving encryption, and future work include expanding our scope to incorporate other anonymization techniques, and studying whether our segmentation-based approach can even be used to hide plaintext data (e.g., the payload). Third, to preserve data utility, our approach imposes a trade-off between privacy and computational cost, so optimizing such a trade-off is also an interesting future topic.

REFERENCES

[1] ISO Std IEC, "ISO 27017," *Information technology- Security techniques (DRAFT)*, 2012.

[2] Cloud Security Alliance, "Cloud control matrix CCM v3.0.1," 2014, available at: https://cloudsecurityalliance.org/research/ccm/.

[3] V. D. Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "A survey of network isolation solutions for multi-tenant data centers." *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2016.

[4] S. E. Coull, C. V. Wright, A. D. Keromytis, F. Monrose, and M. K. Reiter, "Taming the devil: Techniques for evaluating anonymized network data." in *NDSS*, 2008.

[5] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, M. K. Reiter *et al.*, "Playing devil's advocate: Inferring sensitive information from anonymized network traces." in *NDSS*, 2007.

[6] M. Kolhar, M. M. Abu-Alhaj, and S. M. A. El-atty, "Cloud data auditing techniques with a focus on privacy and security," *IEEE Security & Privacy*, vol. 15, no. 1, pp. 42–51, 2017.

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

[8] OpenStack, "Nova network security group changes are not applied to running instances," 2015, available at: https://security.openstack.org/ossa/OSSA-2015-021.html.

[9] J. Fan, J. Xu, and M. H. Ammar, "Crypto-pan: Cryptography-based prefix-preserving anonymization," *CN*, vol. 46, no. 2, 2004.

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

[11] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proceedings of the 22nd ACM SIGSAC*. ACM, 2015.

[12] 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*. Springer, 2005, pp. 179–196.

[13] The Guardian, "Deloitte hit by cyber-attack," 2017, available at: https://www.theguardian.com/business/2017/sep/25/deloitte-hit-by-cyber-attack-revealing-clients-secret-emails.

[14] Wall Street Journal, "Kpmg fires partners," 2016, available at: https://www.wsj.com/articles/u-s-audit-regulator-probing-leak-of-confidential-inspection-information-to-kpmg.

[15] Reuters, "U.S. FCC imposes 25 million fine on AT&T over customer data breach," 2018, available at: https://www.bbc.com/news/technology-32232604.

[16] Egress, "Quarter of UK Employees Have 'Purposefully Leaked Business Data," 2018, available at: https://www.infosecurity-magazine.com/news/quarter-uk-employees-leaked-data/.

[17] MSSPalert, "Gdpr force mssp," 2019, available at: https://www.msspalert.com/cybersecurity-breaches-and-attacks/compliance/gdpr-may-force-mssp-csp-business-model-overhauls/.

[18] ISO Std IEC, "ISO 27002," 2018, available at: http://bcc.portal.gov.bd/sites/default/files/files/bcc.portal.gov.bd/page/adeaf3e5_cc55_4222_8767_f26bcaec3f70/ISO_IEC_27002.pdf.

[19] EU General Data Protection Regulation, "Fines and Penalties," 2018, available at: https://www.gdpreu.org/compliance/fines-and-penalties/.

[20] A. Slagell and W. Yurcik, "Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization," in *SECURECOMM*. IEEE, 2005.

[21] Common Weakness Enumration, "Nova network security group changes are not applied to running instances," 2019, available at: https://cwe.mitre.org/data/definitions/532.html.

[22] OpenStack oraganization, "Openstack," 2017, available at: https://www.openstack.org/.

[23] 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.

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

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

[26] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[27] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *CCS*, 2011.

[28] 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*, 2012, pp. 175–188.

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

[30] 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*. Ieee, 2010, pp. 1–9.

[31] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service cloud computing," in *CCS*. ACM, 2012.

[32] Concordia University, "Segguard extended running example," 2019, available at: http://arc.encs.concordia.ca/demos.html.

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

[34] 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*, 2015, pp. 469–483.

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

[36] OpenStack oraganization, "Horizon," 2018, available at: https://docs.openstack.org/horizon/latest/.

[37] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986.

[38] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis." in *NSDI*, 2013.

[39] 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*, 2014.

[40] L. A. Gunawan and P. Herrmann, "Compositional verification of application-level security properties." in *ESSoS*. Springer, 2013.

[41] S. Majumdar, T. Madi, Y. Wang, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Security compliance auditing of identity and access management in the cloud: application to openstack," in *CloudCom*. IEEE, 2015.

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

[43] S. Kamara, "Encrypted search." *ACM Crossroads*, vol. 21, no. 3, pp. 30–34, 2015.

[44] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data." in *NDSS*, 2015.

[45] O. Pandey and Y. Rouselakis, "Property preserving symmetric encryption," in *ASIACRYPT*, 2012.

[46] M. Burkhart, D. Brauckhoff, M. May, and E. Boschi, "The risk-utility tradeoff for ip address truncation," in *NDA*. ACM, 2008.

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

[48] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 29–38, 2006.

[49] T. Brekne and A. Årnes, "Circumventing ip-address pseudonymization." in *Communications and Computer Networks*, 2005, pp. 43–48.

[50] S. Chatterjee and M. P. L. Das, "Property preserving symmetric encryption revisited," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2015, pp. 658–682.

[51] F. Chen, B. Bruhadeshwar, and A. X. Liu, "Privacy-preserving cross-domain network reachability quantification," in *ICNP*. IEEE, 2011.

[52] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[53] F. McSherry and R. Mahajan, "Differentially-private network trace analysis," *SIGCOMM*, pp. 123–134, 2010.

[54] S. Bleikertz, C. Vogel, and T. Groß, "Cloud Radar: Near real-time detection of security failures in dynamic virtualized infrastructures," in *ACSAC*, 2014.

[55] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese, "Checking beliefs in dynamic networks," in *NSDI'15*, 2015.

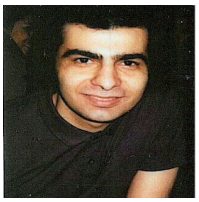
[56] Y. Wang, T. Madi, S. Majumdar, Y. Jarraya, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi, "Tenantguard: Scalable runtime verification of cloud-wide vm-level network isolation," in *NDSS*, 2017.

[57] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: verifying network-wide invariants in real time," in *NSDI*, 2013.

[58] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," in *ICNP*, 2013.

Momen Oqaiy Momen Oqaily is currently working toward the Ph.D. degree in information and systems engineering at the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada. He received his B.S. degree in network engineering and security and the master degree in Information Systems Engineering. Momen research interests include privacy and cloud security auditing.

Dr. Yosr Jarraya Yosr Jarraya is currently a researcher in security at Ericsson. Before that, she had a two-year MITACS postdoctoral fellowship with the company. She was previously Research Associate and Postdoctoral Fellow at Concordia University, Montreal. She received a Ph.D. in Electrical and Computer Engineering from Concordia University. She is the co-author of more than 30 research papers on topics including cloud security, network and software security, formal verification and SDN.



Meisam Mohammady Meisam Mohammady is a PhD candidate at the Concordia Institute for Information Systems Engineering, Concordia University. He got his MSc in Electrical Engineering at Ecole Polytechnique Montreal and his BSc in Electrical and Computer Engineering from Sharif University Of Technology. His research interests include privacy,Cyber Physical Systems and Applied Mathematics.

Dr. Suryadipta Majumdar Suryadipta Majumdar is currently an Assistant Professor in the Information Security and Digital Forensics department at University at Albany - SUNY. Suryadipta received his Ph.D. on cloud security auditing from Concordia University, Canada. His research mainly focuses on cloud security, Software Defined Network (SDN) security and Internet of Things (IoT) security.

Dr. Makan Pourzandu Makan Pourzandu is a research leader at Ericsson, Canada. He received his Ph.D. degree in Computer Science from University of Lyon I Claude Bernard, France and M.Sc. in parallel computing from École Normale Supérieure de Lyon, France. He is the co-inventor of 19 granted US patents and more than 65 research papers in peer-reviewed scientific journals and conferences. His current research interests include security, cloud computing, software security engineering, cluster computing, and component-based methods for secure software development.

Dr. Lingyu Wang Lingyu Wang is a Professor at the Concordia Institute for Information Systems Engineering (CIISE) at Concordia University, Montreal, Canada. He holds the NSERC/Ericsson Senior Industrial Research Chair in SDN/NFV Security. He received his Ph.D. degree in Information Technology in 2006 from George Mason University. His research interests include cloud computing security, SDN/NFV security, security metrics, software security, and privacy. He has co-authored five books, two patents, and over 120 refereed conference and journal articles at reputable venues including TOPS, TIFS, TDSC, TMC, JCS, S&P, CCS, NDSS, ESORICS, PETS, ICDT, etc

Dr. Mourad Debbabi Mourad Debbabi is a Full Professor at the Concordia Institute for Information Systems Engineering and Associate Dean Research and Graduate Studies at the Faculty of Engineering and Computer Science. He holds the NSERC/Hydro-Québec Thales Senior Industrial Research Chair in Smart Grid Security and the Concordia Research Chair Tier I in Information Systems Security. Dr. Debbabi holds Ph.D. and M.Sc. degrees in computer science from Paris-XI Orsay, University, France. He published 3 books and more than 260 peer-reviewed research articles in international journals and conferences on cyber security, cyber forensics, privacy, cryptographic protocols, threat intelligence generation, malware analysis, reverse engineering, specification and verification of safety-critical systems, smart grid, programming languages and type theory.