# On the Universally Composable Security of OpenStack*†

Kyle Hogan‡, Hoda Maleki§, Reza Rahaeimehr§, Ran Canetti¶‖, Marten van Dijk§,

Jason Hennessey¶**, Mayank Varia¶, Haibin Zhang††

August 10, 2018

*The order of the authors is alphabetical among the first three authors, and again alphabetical among the remaining authors. Indeed, the first three authors contributed significantly more to the paper than the other ones.

†This work is supported by the National Science Foundation as part of the MACS Frontier project (bu.edu/macs).

‡Massachusetts Institute of Technology. klhogan@mit.edu.

§University of Connecticut. {hoda.maleki, reza.rahaeimehr, marten.van_dijk}@uconn.edu.

¶Boston University. {canetti, henn, varia}@bu.edu

‖Tel Aviv University. Member of CPIIS. Supported in addition by ISF grant 1523/14.

**NetApp.

††University of Maryland, Baltimore County. hbzhang@umbc.edu.

**Abstract**

We initiate an effort to demonstrate how we can provide a rigorous, perceptible and holistic security analysis of a very large scale system. We choose OpenStack to exemplify our approach. OpenStack is the prevalent open-source, non-proprietary package for managing cloud services and data centers. It is highly complex and consists of multiple inter-related components which are developed by separate, loosely coordinated groups. All of these properties make the security analysis of OpenStack both a crucial mission and a challenging one. We base our modeling and security analysis in the universally composable (UC) security framework, which has been so far used mainly for analyzing security of cryptographic protocols. *Indeed, demonstrating how the UC framework can be used to argue about security-sensitive systems which are mostly non-cryptographic, in nature, is one of the main contributions of this work.*

Our analysis has the following key features:

1. It is *user-centric*: It stresses the security guarantees given to users of the system, in terms of privacy, correctness, and timeliness of the services.
2. It provides *defense in depth*: It considers the security of OpenStack even when some of the components are compromised. This departs from the traditional design approach of OpenStack, which assumes that all services are fully trusted.
3. It is *modular*: It formulates security properties for individual components and uses them to assert security properties of the overall system.
4. It is *extendable*: Due to the scale of OpenStack, we limit the analysis to some core services of OpenStack at a high level. The analysis is extendable to more detail of the services, and other services can be added to the model using the same methodology, without much conceptual difficulty. Because of the modularity of the analysis, new services can be added one by one, almost independently of each other.

Although our analysis covers only a number of core components of OpenStack, it formulates some basic and important security trade offs in the design. It also naturally paves the way to a more comprehensive analysis of OpenStack. In addition, as a by-product result of our modeling, we introduce a novel tokening mechanism, RAFT, which is backward compatible with Fernet Token currently used in OpenStack. By applying the UC framework, we prove that RAFT's one-time use tokens can realize a more secure OpenStack cloud than bearer tokens do.

***Index terms***— Modular Security Analysis, Universal Composability, Cloud Security, OpenStack

# Contents

# 1 Introduction

Analyzing security properties of large-scale information systems is a daunting task. A first challenge, that is almost intractable in and of itself, is to adequately articulate and rigorously express the security requirements of the system in the first place. Indeed, adequately capturing even simple, intuitive concerns is non-trivial. Furthermore, security is often inseparable from the expected functionality, which is complex in of itself. It also invariably has multiple facets and competing requirements that need to be reconciled.

A second challenge is to rigorously assert the specified properties. This challenge is even more daunting, especially when the system consists of multiple components and one has to take into account inter-component interactions, potential failure of individual components, and the associated potential vulnerabilities.

The natural way to deal with such complexities is modularity: formulate and assert the security properties of individual components, and then deduce security properties of the overall, composite system from the security properties of the components, as well as those of the way in which the components are put together. This breaks down the overall analysis into multiple steps where each step deals with a much simpler system. Furthermore, when successful, the analysis would deduce the overall security of the system from the security of the components, and the security of the overall design given the components. Still, breaking down a system to components in a way that allows for effective composable *security* analysis is a non-trivial task in of itself.

A number of frameworks for modular security analysis for cryptographic protocols have been developed over the years, e.g. [1, 2, 3, 4, 5, 6, 7]. Furthermore, a number of works have made advancements to use these frameworks to analyze security of security-sensitive systems that are non-cryptographic in nature, e.g. [8, 9, 10, 11, 12]. Extending these cryptographic frameworks to handle systems with complex interfaces and sizeable codebase is a challenging endeavor — but one that holds great promise. In particular, it opens the door to a rigorous, yet modular and approachable security analysis of large-scale software systems. Indeed, such analysis can be very valuable even in systems that use little or no cryptography.

In this work, we initiate modular security analysis of OpenStack, which is a large-scale distributed system with complex interfaces and use cases. It is also extremely security-sensitive. We perform our analysis within the Universally Composable (UC) security framework.

**OpenStack** OpenStack is a software package for data centers and virtualization services, including remote computation, storage, networking, and related services. It is open-source, and it is developed in a distributed and loosely coordinated way by multiple contributors across the globe. OpenStack is by far the most popular non-proprietary software package of its kind.

The OpenStack project began in 2010 as a collaboration between two groups: Rackspace, a public IaaS provider selling cloud services, and NASA, a part of the United States government that wanted to take advantage of the elasticity and datacenter efficiency benefits that come from combining different workloads into a single private cloud [13]. Since then, it has grown to be a large open source project with over 3.5 million lines of code, and over 6,000 contributors and hundreds of implementations around the world [14, 15]. There is a governance body [16] that actively manages the development and stability of OpenStack.

The design is inherently modular, with 23 modules where each module has some pre-specified functionality, as well as interfaces with the other modules it interacts with. It should be noted

though that the functionality of the interfaces is not completely pinned down; indeed some modules have multiple implementations that provide slightly different functionality. Also many of the modules allow for a variety of underlying software packages as plug-ins. Some of the main modules of OpenStack include:

**Nova (compute):** Manages the creation, maintenance and removal of virtual machines (VMs).

**Glance (image repository):** Stores and manages the images loaded to VMs.

**Cinder & Swift (block & object storage):** Manage the storage of data (in blocks, volumes, and more general objects) for VMs.

**Neutron (networking):** Provides internal and external virtual networks for VMs.

**Keystone (access control and key management):** Holds the permission information controlling the access of users to the services and data. Interacts with users and all other modules to enforce the permission policies.

**Horizon (user-side dashboard):** Provides an interface between users of the system and its service modules.

Each one of these modules is a complex, distributed system in and of itself, sometimes with multiple subdivisions, plug-ins, and alternative implementations.

In terms of security guarantees and analysis, the OpenStack consortium offers an extensive and illuminating security guide [17]. However, this guide concentrates on security measures for protecting the OpenStack package itself from external intruders and attackers. Other literature on the security of OpenStack (e.g. [18, 19, 20, 21, 22, 23]) also concentrate on individual components and mitigation of specific attack vectors. By contrast, to our knowledge we are the first to model and assert the overall security guarantees that the OpenStack services collectively provide to its users.

Furthermore, the design of OpenStack appears to implicitly assume that all components of an OpenStack-based service are trusted. In other words, there is no attempt to guarantee security (at any level) in case that any part of the entire package is compromised or not functioning properly. We discuss this point further below.

**Our analysis** We initiate a study of the security properties provided by OpenStack when viewed as a service to external users which is a typical model for most (large scale) applications. This includes properties such as confidentiality and integrity of *data* (both in storage and in transition), confidentiality and correctness of *computations,* as well as timeliness and resource preservation. We also consider the extent to which these properties are preserved under various attack vectors and when various components of the system are compromised.

We base our analysis in the universally composable security (UC) framework, which provides a way to articulate security properties in a rigorous and precise way. According to the definition of universal composability, a UC-secure component remains secure if it is universally composed with other UC-secure components [3]. The extendability property of universal composability allows us to analyze a part of a system and additively analyze the remaining components. The framework provides a natural and convenient mechanism for arguing about the preservation of security when programs and systems are composed in a modular way. Indeed, from this perspective the UC

framework appears to be ideally suited to analyzing OpenStack whose design is inherently and predominantly modular.

On the other hand, the UC framework was initially created, and predominantly used, for analyzing cryptographic protocols. These are very different than OpenStack: while their analysis requires creative reductions to hard computational problems, they are vastly simpler in terms of number of components, cases, and volume of code. Indeed, coming up with an effective modeling of OpenStack within the UC framework is a labor intensive, non-trivial line of research. This work paves the way in this direction.

Recall that in the UC framework the security requirements from the analyzed system (or, service) $\pi$ are analyzed jointly with the functionality requirements from the service. This is done by way of formulating an *ideal service* $\mathcal{F}$, which specifies the desired response (or lack thereof) to any potential external input. Roughly speaking, the service $\pi$ is said to emulate the ideal service $\mathcal{F}$ if no external environment can tell whether it is interacting with $\pi$ or with $\mathcal{F}$.

In order to account for some level of allowable "slack" for $\pi$ relative to $\mathcal{F}$, the framework allows the analyst to introduce an intermediary, or a *simulator* $S$ that controls some of the interfaces between $\mathcal{F}$ and the environment. That is, service $\pi$ is now said to emulate an ideal service $\mathcal{F}$ if there exists a simulator $S$ such that no external environment can tell whether it is interacting with $\pi$ or with a system where some of its APIs connect to $\mathcal{F}$, and other APIs connect to $S$. (Typically, $S$ connects to APIs that we don't consider to be part of the desired functionality, such as the communication between components of the implementing protocol.)

An attractive property of this definitional style is the following natural security-preserving composability: Since the specification $\mathcal{F}$ is written as an "idealized" service in and of itself, one can design and analyze some other system (or, service) $\rho$ where the components of $\rho$ make calls to one or more instances of the service $\mathcal{F}$. The UC framework guarantees that the protocol $\rho^{\mathcal{F}\to\pi}$, where each instance of $\mathcal{F}$ is replaced by an instance of $\pi$, continues to exhibit the same security and correctness properties as the original protocol $\rho$. In particular, if $\rho$ emulates some other ideal service $\mathcal{G}$, then $\rho^{\mathcal{F}\to\pi}$ will emulate $\mathcal{G}$ just the same. (Note that both $\pi$ and $\rho$ may well be distributed, multi-component systems in and of themselves.)

**This work** Our goal is to demonstrate an approach that enables analysts to analyze the security of OpenStack in a structured and perceptible manner. To do so, we provide initial modeling and analysis of the overall design and operation of OpenStack, as well as the functionality and security requirements from a number of core modules (essentially the modules described above with the exception of Swift and Neutron). Our analysis validates the overall security of the design, while at the same time formulating some security weaknesses. Although, the weaknesses are conceptually known to the OpenStack community, our analysis shows the right level at which these issues must be dealt. For example, Sze et al. [24] tried to solve the token problem by assuming a trusted component inside Nova. Our analysis shows that such designs are not a suitable design decision if we are looking for a UC-secure system. We also propose and analyze methods for properly overcoming these weaknesses.

We stress that our analysis only covers the high level design of some main components of OpenStack. A detailed analysis of vulnerabilities within such a huge system are not the focus in this paper; nevertheless, our work does pave the way for future work to model and analyze additional components, or to analyze the structure and security of individual components in more detail.

We first formulate an ideal cloud $\mathcal{F}_{Cloud}$ that provides a simple specification of the functionality and security that we assert OpenStack achieves. This formulation naturally involves many design choices and parameters that affect the security and functionality requirements imposed on the system. We discuss them within. One important aspect of our ideal cloud specification is the expected behavior upon various types of partial corruption (which correspond to corruption of individual modules in an OpenStack service). This is where we depart from the current OpenStack package, which does not provide any security guarantees as soon as any module is corrupted.

Next we formulate ideal functionalities that correspond to the four services we capture, namely $\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, $\mathcal{F}_{BlockStorage}$, and $\mathcal{F}_{Identity}$. Our models for each OpenStack service aim at capturing the functionality and intricacies of the actual components of OpenStack, modulo some necessary modifications that are essential for security. Also here we face a number of choices that represent different levels of security of these services.

These services communicate with each other via secure message transmission $\mathcal{F}_{SMT}$. Additionally, they use an external network $\mathcal{F}_{ExtNet}$ to communicate with the user, or more specifically to connect to the user's *Dashboard* program (which is our abstraction of Horizon). Collectively, the joint interactive effort of these services and protocols comprise a cloud of *OpenStack Services*. In the two main results of our paper (Theorems 2 and 3), we prove that the OpenStack services collectively UC-realize our ideal cloud.

**Security weaknesses formalized** As long as all the components of an OpenStack service continue to function properly, the Open Stack design indeed provides adequate security, namely secrecy and correctness of data and computations.

At the same time, our analysis formally shows the extent to which OpenStack is vulnerable to the compromise of a subset of components. We remark that the case where some components become compromised is quite realistic; indeed, some components, such as the VM manager Nova, are more susceptible to attack just because they are exposed to a richer attack surface from malicious VMs. Consequently it is prudent to design the system so as to minimize the damage from the compromise of individual components, and to perform analysis that provides some security guarantees even in case that some components are adversarially controlled.

We consider several sources of weakness in the OpenStack design that get uncovered by our analysis. For one of these weaknesses we also propose a solution and demonstrate that the solution provides a tangible increase in security. (We stress that neither of this issues is new; still our analysis puts these issues in context and points to potential solutions.)

The first issue we discuss is the *bearer token* mechanism to authenticate users and verify their authorization to access resources. Bearer tokens operate as follows. When user $U$ wishes to launch a VM, it first contacts the identity service Keystone, and obtains a token that represents for Keystone the user $U$ and its capabilities. $U$ then hands the token to the compute service Nova, which starts a VM for $U$ with the desired parameters, and then *continues to use the token on behalf of* $U$ in order to obtain additional services for $U$'s VM (or VMs) as becomes necessary. For instance, Nova may use the token on behalf of $U$ to obtain storage or networking services for the VM. (A natural alternative to the token mechanism is a digital signature by Keystone regarding the capabilities of $U$. However, this solution has been rejected by the OpenStack community due to its computational and bandwidth overhead.)

As long as the communication between services is secured via point-to-point secure session protocols (say, via TLS), this mechanism provides security against external attackers that only

control the network. However, this mechanism allows a corrupted component (say, Nova) to impersonate tokens on behalf of any user. When the inter-process communication is not secured in a point-to-point way, we have that *any* rogue OpenStack entity that can eavesdrop to the inter-service communication (say, a hypervisor that was compromised by its tenant VM) can potentially have access to all current bearer tokens in the system. Indeed, previous works (e.g. [25]) have already pointed out this weakness and proposed limiting the scope of these bearer tokens by setting expiration times and other scoping mechanisms.[1]

It should be stressed that, upon each new use of the token, each new service verifies the token again with Keystone. Ergo, tokens that are invalid will not cause damage. However, when the tokens are broadly scoped, nothing prevents a rogue component from using legitimate tokens of existing unsuspecting users to compromise both the integrity and the secrecy of their data.

This work analyses OpenStack with two token mechanisms. First, we analyze OpenStack's existing bearer tokens. Our analysis formulates that the current OpenStack realizes an "ideal cloud" specification that provides little security as soon as any component is corrupted.

We then construct a new, stronger *one-time* token mechanism that is a more stringent variant of the mechanism proposed in [25], and we prove that this mechanism suffices for realizing an ideal cloud that provides some meaningful security properties even when some of the services are fully corrupted. In a nutshell, our mechanism has Keystone generate a token that is authenticated via a cryptographic authentication tag, using Keystone's own master secret key. A token-specific key is then derived and sent to the user's Horizon module over a secure communication channel. The user then scopes the token for a *single use* by a specific set of services (e.g., starts a specific VM with a specific image and a specific amount of block storage attached), authenticates the scoped token with the derived key it received from Keystone, and sends the derived token to the appropriate service (say, Nova). Nova then contacts Keystone to verify the token before honoring it. In addition, Nova can scope down and pass the token to other services as needed (say, Glance or Cinder); this is done over authenticated channels. Each service *records the source of the token*, and it interacts with Keystone to verify the token *along with the service from which the token arrived*.

We show that the additional security provided by the limitation to one-time use, together with the ability to identify the entity that provides the token, suffices for realizing a significantly stronger variant of the ideal cloud specification that limits the damage caused by corrupted services. Allowing the Horizon module to scope the token enables the fine-grained scoping of the one-time token. Furthermore, the Horizon module of a user is now capable of generating multiple one-time tokens on its own, without intervention of Keystone.

Our basic modeling represents a VM as a new process that runs independently of the core Nova component and all other VMs. This means that a VM continues to execute unmodified *even when Nova is corrupted.* This is a strong guarantee, which the current OpenStack package does not meet (for instance, a hypervisor that's corrupted by its tenant VM has access to all the communication between components on the main communication bus). To capture the properties of the existing design we formulate an additional corruption mechanism whereby the ideal cloud no longer provides any secrecy or integrity guarantees as soon as any component is corrupted. (Still, this modeling makes it possible to assert stronger security properties for VMs, when employing appropriate mechanisms, such as homomorphic encryption or other cryptographic protection mechanisms.)

[1]For simplicity of exposition we leave the timeout mechanism (as well as measurement of real time) outside the model. We note that timing mechanisms can be added in a relatively straightforward way, using the UC-style modeling of network time of Canetti et al. [26]. Indeed, the ability to modularly add the consideration of time is another demonstration of the power of composable security analysis.

**Towards mechanized analysis** One of the most important aspect of this work, that sets it apart from many previous works in the UC framework, is that we provide in full detail the specifications of the ideal cloud and the individual services, as well as the descriptions of the simulators and the proofs of security, without glossing over steps. As a consequence, our proofs and specifications are decently long and tedious. Indeed, while for this paper we stick to pen-and-paper proofs, we believe that our modeling and analysis are readily amenable to mechanization, and also to some level of automation. Natural candidates for tools that would enable such mechanized analysis include the EasyCrypt tool [27], the FCF tool [28], or the CryptHOL tool [29].

### 1.1 Related Work

**Security Analysis of Clouds** The OpenStack Security Guide [17] goes into depth about the security of different aspects of configuring the many different pieces of OpenStack. However, it does not provide any security analysis, formal or otherwise, nor does it consider situations where a cloud service is compromised.

Other works have focused on the compromise of compute nodes [24] or parts of the management infrastructure [30], and Sun et al. [31, 32, 33] specifically discuss limiting the scope of compromised OpenStack services. These works conclude that corrupted cloud components have far-reaching security impact and can in many cases compromise the privacy and integrity of all cloud operations. Their conclusions highlight the need for a formal security analysis of service corruptions in OpenStack, which we provide with our construction. Sze et al. [24] additionally propose an alternative authorization tokening mechanism to reduce the effect of corrupted compute nodes, but their construction neither protects against compromise of other services nor provides token authentication and replay prevention. We have focused on addressing these requirements as well as shifting control of token generation and scope to the user responsible for the request. Also, crucially, we provide a security analysis that concretely specifies the security gain.

**Using UC** Canetti et al. [34] show how the UC framework can be used to analyze the simple components of a file system in isolation and to guarantee that these components maintain their behavior in the larger system even under adversarial conditions. This demonstrates basic integrity properties of the file system, i.e., the binding of files to filenames and writing capabilities. Gajek et al. [35] evaluate in the UC framework the emulation of secure communication sessions by the composition of key exchange functionalities that are realized by the TLS handshake and record layer protocols. Canetti et al. [36] give a modular and global universally composable analytical framework for PKI-based message authentication and key exchange protocols.

For our analysis, we apply the style of [34] to the larger and more complex OpenStack framework and utilize aspects of [35, 36] to achieve secure communication. We further use our construction to demonstrate security flaws in OpenStack's current authorization mechanism and assess the improvements provided by our suggested changes.

**Alternative Formalisms** The UC framework is not the only option for formal analysis of computing systems. In particular, Gu et al. [10] use the Coq proof assistant to analyze and provide an abstraction of layers of the computing stack including the kernel, networking, etc. They developed and verified a certified kernel with 37 of these abstraction layers.

We chose to use UC for our analysis because its modularity and composability aligned well with the structure of OpenStack which is itself composed of many services that interact via a series of

well defined APIs. These services support varying interchangeable implementations that would be difficult to support using a less modular proof framework.

### 1.2 Organization

The remainder of this work is organized as follows. We begin in Section 2 by providing some background on OpenStack along with an overview of the UC framework. Next, Section 3 provides an informal account of our modeling of OpenStack services and the security properties we chose to model. We use an example workflow to highlight the modularity of our design. In Section 4, we fully specify the functionalities for the OpenStack services and an idealized monolithic cloud, and then we prove that the former UC-realizes the latter. Then, we present a stronger token mechanism in Section 5, and we perform another UC analysis in Section 6 to prove that our new tokens improve the security of OpenStack in the event that some services are compromised. Finally, we conclude with a discussion of future work in Section 7.

## 2 Background

In the following subsections, we focus on surveying OpenStack and briefing UC.

### 2.1 OpenStack

As outlined in the Introduction, OpenStack is a modular, distributed, open-source cloud computing software stack for providing Infrastructure as a Service (IaaS) to multiple (potentially untrusting) users. In this section, we describe OpenStack's operation with a focus on some of the security concerns of its authorization system.

**Modular services** OpenStack comprises several distinct *services* that offer different features like Virtual Machine (VM) computation, object storage, identity authentication and authorization, block data storage, networking, and VM image management. Many services act primarily as orchestration engines, exposing the functionality of a rich ecosystem of plugins through the service APIs [37]. For example, the VM service (called Nova) has plugins supporting different hypervisors, including Microsoft HyperV, Xen and VMware. Services are independent, differing both in development, where services have separate teams, and in deployment by *cloud providers*, where services can be managed and scaled separately.

OpenStack's identity service, named Keystone, links these independent services by serving as a common method for authenticating and authorizing users to the different services. Keystone codifies permissions for users based upon the *projects* they may access and the *roles* they have on these projects. A project is simply a collection of resources (VMs, networks, object storage, etc) that a group of authorized users may access. Roles describe the actions a user may take within each project. While OpenStack supports a robust set of roles, in practice most deployments only configure two: an ordinary *member* and an *administrator* with additional power.

**Trust model** Many of OpenStack's design choices and security issues stem from its broad trust model, which assumes that all services act as faithful user agents. Providing security even in the case where some services are comporomized does not appear to be a design goal. Furthermore,

interactions between services in OpenStack are optimized in light of this trust. However, OpenStack's unprotected interior means that a (partially) compromised service can do a great deal of harm: acquiring a single bearer token allows the compromised service to impersonate the user for any subsequent action.

**Tokens** To determine a user's project and role, Keystone gives the user a *bearer token* after authenticating with their credentials (e.g., username and password); users include this token in API requests to other services for authentication and authorization. Services pass this token to Keystone, which returns back a *(project,role)* tuple if the token was valid. Services then make all authorization decisions based on that tuple. Bearer tokens are used similarly in other popular protocols, like OAuth [38]. Because possession of a bearer tokens grants access to resources, care must be taken to protect them from unauthorized parties using methods like TLS [17].

Service interactions on behalf of an end user allow for more complicated tasks such as attaching storage volumes to a compute node. The compute service is able to send a user's token to the storage service which can in turn verify with the identity service that this token has access to the requested volume and attach it to a node without needing to check with the end user itself.

OpenStack uses plugins to Keystone to implement tokens, the most popular being UUID and Fernet [39]. UUID tokens issue random, universally unique identifiers [40] to users after a successful first authentication with Keystone, and stores them in a database with other required information such as expiration time, the project and role associated with it.

The Fernet token is a recent innovation that uses cryptography to provide authenticity without accessing a central DB. It is a mechanism by which keystone creates a private, authenticated channel to itself. It has quickly become the preferred token format for OpenStack as they do not require maintaining a central database of valid tokens, which adds network load and latency.

## 2.2 Universally Composable Security

Here, we provide a brief overview of the UC framework. See [3] and [41] for more details. (This overview is taken almost verbatim from [36].)

We focus on the notion of protocol *emulation*, wherein the objective of a protocol $\pi$ is to imitate another protocol $\phi$. In this work, the entities and protocols we consider are polynomial-time bounded Interactive Turing Machines (ITMs), in the sense detailed in [3].

### 2.2.1 Systems of ITMs

To capture the mechanics of computation and communication among entities, the UC framework employs an extension of the ITM model. A computer program (such as for a protocol, or perhaps program of the adversary) is modeled in the form of an ITM. An execution experiment consists of a system of ITMs which are instantiated and executed, with multiple instances possibly sharing the same ITM code. A particular executing ITM instance running in the network is referred to as an ITI. Individual ITIs are parameterized by the program code of the ITM they instantiate, a party ID (pid) and a session ID (sid). We require that each ITI can be uniquely identified by the identity pair id = (pid,sid), irrespective of the code it may be running. All ITIs running with the same code and session ID are said to be a part of the same protocol session, and the party IDs are used to distinguish among the various ITIs participating in a particular protocol session.

### 2.2.2 The Basic UC Framework

At a very high level, the intuition behind security in the basic UC framework is that any adversary $\mathcal{A}$ attacking a protocol $\pi$ should learn no more information than could have been obtained via the use of a simulator $S$ attacking protocol $\phi$. Furthermore, we would like this guarantee to hold even if $\phi$ were to be used as a subroutine in arbitrary other protocols that may be running concurrently in the networked environment and after we substitute $\pi$ for $\phi$ in all the instances where it is invoked. This requirement is captured by a challenge to distinguish between actual attacks on protocol $\phi$ and simulated attacks on protocol $\pi$. In the model, attacks are executed by an environment $E$ (Env) that also controls the inputs and outputs to the parties running the challenge protocol. $E$ is *constrained* to execute only a single instance of the challenge protocol. In addition, it is allowed to interact freely with the attacker (without knowing whether it is $\mathcal{A}$ or $E$). At the end of the experiment, the environment is tasked with distinguishing between adversarial attacks perpetrated by $\mathcal{A}$ on the challenge protocol $\pi$, and attack simulations conducted by $S$ with protocol $\phi$ acting as the challenge protocol instead. If no environment can successfully distinguish these two possible scenarios, then protocol $\pi$ is said to *UC-emulate* the protocol $\phi$. We state the formal definition:[2]

**Definition 1** (UC-emulation)**.** *Let $\pi$ and $\phi$ be multi-party protocols. We say that $\pi$ UC-emulates $\phi$ if for any adversary $\mathcal{A}$ there exists a simulator $S$ such that for any environment $E$, we have:*

$$\text{EXEC}_{\pi,\mathcal{A},E} \approx \text{EXEC}_{\phi,S,E}$$

Defining protocol execution this way is sufficient to capture the entire range of network activity that is observable by the challenge protocol but may be under adversarial control.

### 2.2.3 Univeral composition

the UC framework admits a very strong composition theorem, which guarantees that arbitrary instances of $\phi$ that may be running in the network can be safely substituted with any protocol $\pi$ that UC-emulates it. For simplicity, we first restrict attention to a special class of protocols, called subroutine-respecting protocols. Essentially, these are protocols that preserve a "hierarchical" structure of subroutine calls.

**Definition 2** (Subroutine-respecting protocols)**.** *We say that a protocol $\pi$ is* subroutine-respecting *if the following properties hold with respect to every instance of $\pi$ in any execution of any protocol $\rho$ that makes subroutine calls to $\pi$:*

1. *No ITI which is a subsidiary of this instance passes inputs or outputs to an ITI which is not a party or subsidiary of this instance.*
2. *At first activation, each ITI that is currently a subsidiary of this instance, or will ever become one, sends a special message to the adversary, notifying it of its own code and identity, as well as the code $\pi$ and session ID of this instance. We call this requirement* subroutine publicness*.*

**Theorem 1** (UC-Composition)**.** *Let $\rho$,$\pi$ and $\phi$ be protocols such that $\rho$ makes subroutine calls to $\phi$. If $\pi$ UC-emulates $\phi$ and both $\pi$ and $\phi$ are subroutine-respecting, then protocol $\rho^{\phi \Rightarrow \pi}$ UC-emulates protocol $\rho$.*

[2] We consider only polynomial-time environments and adversaries. We omit from this overview the precise definitions of polytime and balanced environments, as they are inconsequential for this work.

Here $\rho^{\phi \Rightarrow \pi}$ denotes the composed protocol, i.e. protocol $\rho$ where each call to an instance of $\phi$ is replaced by a call to an instance of $\pi$ with the same session ID.

#### 2.2.4 The Generalized UC Framework.

As mentioned above, the environment $E$ in the basic UC experiment is unable to invoke protocols that share state in any way with the challenge protocol. In many scenarios, the challenge protocol produces information that is shared by other network protocol sessions. For example, protocols may share information via a global setup such as a public Common Reference String (CRS) or a standard Public Key Infrastructure (PKI). The basic UC framework discussed above does not address this kind of shared state; moreover, the UC composition theorem does not hold for non-subroutine-respecting protocols (i.e., protocols that share state information with other protocol sessions). Still, we would like to analyze such protocols in a modular way. To overcome this limitation, [41] proposes the Generalized UC (GUC) framework. The GUC challenge experiment is similar to the basic UC experiment, only with an *unconstrained* environment. In particular, now $E$ is allowed to invoke and interact with arbitrary protocols, and even multiple sessions of the challenge protocol. Some of the protocol sessions invoked by $E$ may even share state information with challenge protocol sessions, and indeed, those protocol sessions might provide $E$ with information related to the challenge protocol instances that it would have been unable to obtain otherwise. To distinguish this from the basic UC experiment, we denote the output of an unconstrained environment $E$, running with an adversary $\mathcal{A}$ and a challenge protocol $\pi$ in the GUC protocol execution experiment, by $\text{GEXEC}_{\pi,\mathcal{A},E}$. GUC emulation is defined analogously to the definition of basic UC emulation outlined above:

**Definition 3** (GUC-emulation)**.** *Let $\pi$ and $\phi$ be multi-party protocols. We say that $\pi$ GUC-emulates $\phi$ if for any adversary $\mathcal{A}$ there exists an adversary $S$ such that for any (unconstrained) environment $E$, we have:*

$$\text{GEXEC}_{\pi,\mathcal{A},E} \approx \text{GEXEC}_{\phi,S,E}.$$

## 3 Modeling the Security of OpenStack: Informal Overview

In this section, we provide an informal account of our modeling of OpenStack and the security guarantees we assert. We first describe the behavior of each service and the risk associated with its compromise (Section 3.1). Then, we generalize from the service-level issues to provide informal, holistic security properties about OpenStack as a whole (Section 3.2). Next, we survey the design decisions and degrees of freedom that influence our model (Section 3.3). The informal account in this section is then followed by the actual definitions of the ideal cloud (Section 3.4) and the OpenStack services (Section 3.5).

Following the approach of the UC framework, we consider an adversarial environment $E$ that controls all the interfaces of the legitimate users with the analyzed service, and in addition controls the communication network and the compromised components of the system.

In the context of our OpenStack service, this means that $E$ can create new compute nodes with specific images of its choice, and link nodes to storage volumes subject to their capabilities. In addition, $E$ can delay or drop arbitrary traffic on the external network (e.g., the Internet) over which users communicate with OpenStack. Next,$E$ can compromise one or more OpenStack services, and thus we reinforce the services to provide defense-in-depth against service-level compromise. We consider both passive corruptions in which the compromised services continue to function normally

but only leak their internal states to $E$, and complete corruptions where the compromized services start running code provided by $E$.

It is stressed that, while the modeling and analysis considers only the interaction between $E$ and a single instance of our cloud service, the universal composition theorem guarantees that the same security guarantees continue to hold even when $E$ is interacting concurrently with other instances of our system and with arbitrary other systems.

### 3.1 Functionality and Security of Each Service

We begin by describing several functionalities that encapsulate both the functionality and security relationships between the OpenStack services and the user's dashboard protocol. In particular, we model the following functionalities in this work:

**Dashboard** Unlike the services described below, the Dashboard protocol is owned and operated by a single user. The Dashboard specifies the sequence of service requests needed to satisfy the user's desires.

Compromising either the Dashboard or the user directly gives $E$ the user's credentials. Hence, $E$ can execute any operation that the user has privileges to perform, but cannot otherwise tamper with the services in any way; in particular, users never learn each other's credentials.

**Identity** $\mathcal{F}_{Identity}$ is responsible for managing credentials. It communicates with all users and services. We presume that $\mathcal{F}_{Identity}$ is instantiated with credentials for each user and service; in practice, these credentials correspond to bearer tokens that can be acquired via an authentication protocol involving a username/password. Subsequently, when any service $\mathcal{F}_{Service}$ receives a request, it may ask $\mathcal{F}_{Identity}$ to validate whether the request is authorized based upon the credentials provided. Additionally, note that while OpenStack uses a project/role based permissions system, our modeling is agnostic to the design of credentials.

When $E$ compromises $\mathcal{F}_{Identity}$ essentially has full control of OpenStack. It immediately acquires the credentials of all users and can even change the permissions associated with them. Furthermore, because all services outsource their authorization decisions to $\mathcal{F}_{Identity}$, $E$ can make any request and convince all services to execute it.

**Compute** $\mathcal{F}_{Compute}$ is responsible for managing the computing nodes on the cloud. It expects that the commands it receives over the network originate with the user's dashboard service. Then, it relies upon the other OpenStack services to aid in fulfilling these requests. In more detail, $\mathcal{F}_{Compute}$ accepts commands from users to create, access, or delete computing nodes. In response, it may request images from $\mathcal{F}_{Image}$, connect to volumes stored on $\mathcal{F}_{BlockStorage}$.

Compromising $\mathcal{F}_{Compute}$ gives the environment extensive power: it may create or delete arbitrary nodes from $\mathcal{F}_{Compute}$'s records and may also capture the credentials of any user who subsequently accesses the service and use these credentials to falsify requests to other services. However, nodes already in existence enjoy a type of forward secrecy and integrity guarantee: once spawned honestly, they run independently of $\mathcal{F}_{Compute}$ and thus are not affected by its future corruption.

**Image** $\mathcal{F}_{Image}$ stores virtual machine images that can be used when instantiating new nodes. These images may either be publicly accessible, or restricted only to users in the appropriate

project. It only provides one method that $\mathcal{F}_{Compute}$ may invoke to request an image. $\mathcal{F}_{Image}$ will respond as long as credentials with appropriate permissions are provided.

Compromising $\mathcal{F}_{Image}$ allows the environment to learn both the images stored on the service as well as all user credentials that pass through it. However, a compromised $\mathcal{F}_{Image}$ cannot directly influence other services since they never expect incoming connections directly from $\mathcal{F}_{Image}$.

**Node** $\mathcal{F}_{Node}$ is our abstraction of a virtual machine; it can execute arbitrary programs on behalf of the project that owns it. Nodes are spawned by $\mathcal{F}_{Compute}$ but then act independently. Because $\mathcal{F}_{Compute}$ sends the code of a node over the network when it is instantiated, $E$ may view all code executed within each node.

Compromising $\mathcal{F}_{Node}$ gives the environment the ability to maul the computation performed within the node.

**Storage and volumes** $\mathcal{F}_{BlockStorage}$ manages the collection of data volumes available for use by users. It provisions volumes and attaches them to nodes, but then is out of the loop during subsequent data accesses.

Compromising $\mathcal{F}_{BlockStorage}$ permits the environment to attach and detach volumes from nodes of her choice. As a countermeasure to protect the data from unauthorized disclosure, the volume can be encrypted with a key that is only known to users with the correct project permissions.

**Message bus** OpenStack has an internal message queue to handle communication between services. It allows us to optionally enable TLS for the inter-service communication. We model the TLS-enabled message bus using a secure message transmission functionality $\mathcal{F}_{SMT}$ that protects the integrity of messages; additionally, it protects the confidentiality of tokens. This is a deviation from OpenStack as-is, which would allow any compromised service to breach message integrity and token confidentiality [21].

External communication between services and users (or their Dashboards) is handled instead by $\mathcal{F}_{ExtNet}$, which provides data confidentiality but does not authenticate the message's sender. This modeling decision reflects the fact that OpenStack never verifies whether the user sending the message is actually the owner of the credentials contained therein.

### 3.2 Security Assertions

We list below several security guarantees. We stress that this is an informal description of forbidden or 'blacklisted' activities; the UC modeling of Sections 3.4 and 3.5 specifies exactly the set of permissible activities in a 'whitelist' format.

A main ingredient in our modeling and analysis is the behavior of the ideal cloud upon corruption of individual services. This way, we capture the compromises we consider and the security properties we guarantee in face of compromise.

**Authentication & authorization** As long as $\mathcal{F}_{Identity}$ is uncompromised, $E$ is limited to perform only those actions authorized by her projects and roles. Corrupt services can perform actions within their scope on behalf of the environment, but cannot influence uncorrupted services to perform unauthorized actions.

**User control** By moving away from bearer tokens, we can provide some user control even in the face of service-level compromises. Bearer tokens allow a corrupted service to impersonate a user to other, uncorrupted services and perform unintended actions. See Section 5 for details on our new tokening mechanism that removes the ability to replay user tokens and thus reduces the scope of a corrupted service to only those actions the service is able to perform directly. As $\mathcal{F}_{Node}$ does not have access to user tokens, a corrupted $\mathcal{F}_{Node}$ is unable to make changes affecting the OpenStack control or data plane. In this sense we model $\mathcal{F}_{Node}$ as being fully isolated from other $\mathcal{F}_{Node}$ instances or OpenStack services. Future expansion of our $\mathcal{F}_{Compute}$ model could include a hypervisor-like functionality detailing this isolation of $\mathcal{F}_{Node}$.

**Resource control** Users may restrict the environment from accessing and tampering with computing nodes, data volumes, and images as long as two services remain uncompromised: $\mathcal{F}_{Identity}$ and the service managing the object. Put simply, the services properly separate their control and data planes. For example, a corrupted compute can delete arbitrary user nodes, but it cannot influence the data stored on unattached volumes or the actions of other nodes (e.g., request a new image from the image service) without user authorization. This guarantee holds only if $E$ does not legitimately hold the required project/role permissions.

Note that all of the guarantees described above only apply at the OpenStack layer. For instance: if you use OpenStack to spawn a web server with several known vulnerabilities and then connect it to the Internet, it is certainly possible for $E$ to compromise your node. We make no guarantees about the safety of objects stored *within* OpenStack, only about their management *by* OpenStack.

Additionally, enforcing these security guarantees may come at the expense of flexibility. Having all security at the border and full trust within OpenStack makes it easier to realize the cloud vision of fungibility; for instance, if one node fails then any worker can be tasked automatically to take over for it. By chaining all authorization decisions back to the user, we reduce the cloud's ability to self-regulate load balancing, scaling, and failover decisions.

### 3.3 Modeling Decisions

In this section, we discuss some of the decisions that impacted our modeling. First, we needed to decide the scope of $\mathcal{F}_{Compute}$ within Nova, the largest OpenStack service. At a high level, Nova comprises both the front-end API/scheduler and the back-end worker nodes. We choose to be more fine-grained so that our model is capable of describing the effects of compromising part, but not all of the (large) Nova code-base. This decision is made without loss of generality; compromising the entire Nova service corresponds in our mode to corrupting $\mathcal{F}_{Compute}$ and all $\mathcal{F}_{Node}$ functionalities. Second, we augment $\mathcal{F}_{Identity}$ in Section 5 to strengthen tokens so they aren't susceptible to data spills. Third, $\mathcal{F}_{SMT}$ assumes that services register keys with the message queue so that it can enforce data integrity and token confidentiality in transit on the internal network.

### 3.4 The Ideal Cloud

Our ideal cloud functionality is a UC functionality that provides the user with the following set of commands:

**CreateNode** Allows a user to create a new node.

**Algorithm 1** Simplified Ideal Cloud (full version in Algorithm 8)

```
1: Upon receiving (Receiver, "Delete Node", session-id, node-id) from E:                ▷ Step 1
2: Send-Sim (Receiver, "Delete Node", session-id, user-id, node-id);                    ▷ Step 2
3:
4: Upon receiving ("Confirm", session-id) from Sim:                                     ▷ Step 3
5: if FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id) then
6:     if user-id is valid & user-id is allowed to delete node node-id then
7:         Valid=1;
8:     else
9:         Valid=0;
10:    end if
11:    Send-Sim (Receiver, "Delete Node", session-id, user-id, node-id, Valid);         ▷ Step 4
12: end if
13: Upon receiving ("Delete Node", session-id, Continue) from S:                        ▷ Step 5
14: NodeExist = 0;
15: if there is node with id=node-id & Valid=1 then
16:    NodeExist = 1;
17:    Delete node-id from the Node list;
18: end if
19: Send-Sim ("Delete Node Completed", session-id, Valid, NodeExist);                   ▷ Step 6
20:
21: Upon receiving ("Output Delete Node", session-id) from S:                           ▷ Step 7
22: if NodeExist=0 or Valid=0 then
23:    Output("Delete Node", session-id, node-id, Fail) to E;                           ▷ Step 8
24: else
25:    Output("Delete Node", session-id, node-id, Success) to E;                        ▷ Step 8
26: end if
```

**DeleteNode** Allows a user to delete a node that they had previously created.

**AccessNode** Allows a user to execute a command on one of their nodes.

**AttachVolume** Allows a user to attach one of their volumes to an existing node.

**DetachVolume** Allows a user to detach a volume that had been attached to one of their nodes.

It is, in a sense, the simplest specification that is faithful in both functionality and security to the real services. As with all UC functionalities, the simplicity of the ideal cloud is intended to promote understanding and transparency of OpenStack's behavior.

Here, we exemplify a simplified version of ideal cloud functionality $\mathcal{F}_{Cloud}$ for the Delete Node function in Algorithm 1. (The full version of this algorithm, which includes message buffering, is written in Algorithm 8.) Our formulation of $\mathcal{F}_{Cloud}$ is simple: $\mathcal{F}_{Cloud}$ asks the permission of the simulator $S$ for receiving every Delete Node request, and also its permission for sending each notification back to the environment. Also, $\mathcal{F}_{Cloud}$ does not hide the user credential validity information and the node existence information, as the adversary may discover the information from the execution of requests.

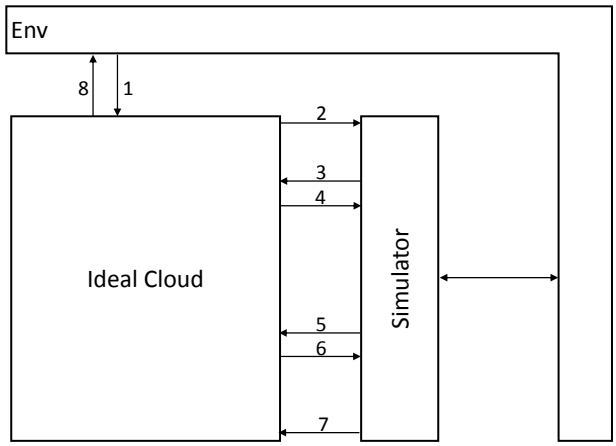


Figure 1: Delete Node in Ideal Cloud with the Simulator.

The information sent from the ideal cloud to the ideal-model adversary (i.e., to the simulator) represents the information that's allowed to be leaked. Specifically, we hide the user and service credentials from the adversary, but leak all other information. We show that, even with this advantage, the ideal cloud guarantees that the adversary cannot impersonate a user and make requests on their behalf without compromising the user or services.

The ideal cloud also captures the security guarantees that are still provided in case some of the services get corrupted. In particular, the ideal cloud specifies the allowed degradation in security when a particular service is corrupted. (Notice that in the context of the ideal cloud the various services are merely names, or tags for the corruption operation made by the adversary.)

### 3.4.1 Ideal Cloud Walkthrough

In the ideal cloud setting, since we model the cloud as a single entity, there is no internal communication and it is left to the simulator to provide the necessary interaction with the environment. The DeleteNode request begins at Step 1 (Figure 1) when the environment sends a Delete Node message to the ideal cloud, through a dummy user who simply forwards inputs. The ideal cloud, could simply, check that the indicated user had the correct permissions to delete the requested node and, if so, removes it from the list of active nodes. However, to capture the fact that the system leaks whether a node was deleted, the ideal cloud notifies the simulator when a node is deleted. When the cloud receives the confirmation message from $S$ in Step 3 it verifies that the node exists and that user has permission to delete it and relays this information to the simulator in Step 4. In Step 5, the simulator tells the cloud to continue. The cloud will then remove the node from the list of active nodes and notifies $S$ in step 6. By receiving the continue message from $S$ (step 7), the cloud outputs the success message to the environment through the dummy user in Step 8.

The simulator acts somewhat differently if a service (say, the Nova compute service) has been compromised. For this reason, the ideal cloud sends $S$ a list of user-ids that have been compromised when compute is corrupted. Additionally, we observe that the environment can send any message on behalf of the corrupted compute; since the simulator cannot directly answer any requests that the environment might make to another (uncompromised) service, $S$ must forward such requests to the ideal cloud and get a response through its specific interface. This decreases the security guarantees that are provided by the cloud.

### 3.4.2 Accounting for Existing Weaknesses

In the case of having compromised services, in order to UC-emulate OpenStack Services, we had to weaken the security guarantees of the ideal cloud. For example, when Nova is compromised, the adversary is able to send a request to Glance using user's credential to get an image. This means that the simulator should be able to provide the requested image to the environment. The simulator does not have the image, therefore it need to ask the ideal cloud. However, the ideal cloud does not respond to this type of simulator's requests. In order to realize the OpenStack Services, we had to remove some of ideal cloud security check points, which decreases the security guarantees. That is, for this example, when the simulator sends a request to get the image (for a corrupted user) while the user has not requested a node to be created, the ideal cloud does not check whether the user has made a "create node" request or not and instead will simply respond to the simulator.

The OpenStack security imperfections discussed in the introduction under the subtitle *Security weaknesses formalized* are the principal reasons for decreasing the security guarantees.

In order to understand why these weaknesses come into place, in the next section, we look into how real services work. We will also explore one of these weaknesses (the token mechanism) in more detail and see how an improved version yields a stronger ideal cloud (Section 5).

## 3.5 OpenStack Services

In this section we describe our model of a simplified OpenStack cloud. The service functionalities, in conjunction with the message passing functionalities, collectively provide the same set of possible commands as the ideal cloud in the previous section.

We require both that the services maintain the confidentiality of users' credentials while executing the commands and enforce that only a user in possession of the required credentials will be

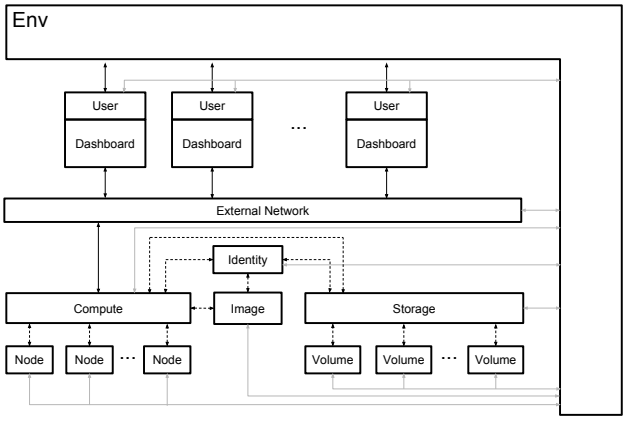


Figure 2: OpenStack Services: Arrows indicated expected lines of communication; other communication flow is possible, but will be ignored. Gray arrows show communication with the adversary, dashed black arrows indicate communication through $\mathcal{F}_{SMT}$, and solid black arrows indicate a direct communication.

able to execute a command.

### 3.5.1 Example Workflow

We will walk through an example of how a user would delete their node using these service functionalities and a dashboard protocol. This example will show the interaction between the $\mathcal{F}_{Compute}$ and $\mathcal{F}_{Identity}$ services and the user. We note that all messages between services will pass through $\mathcal{F}_{SMT}$ while any messages between the user and services will go through $\mathcal{F}_{ExtNet}$. We will briefly mention this during the description and it can be seen in Figure 3 as well.

This interaction begins at Step 1 (Figure 3, Algorithm 2) when the environment sends a DeleteNode message of the form ("Delete Node", session-id, node-id) to the user. The user will then reformat this message in Step 2 to include their credentials and send it to $\mathcal{F}_{ExtNet}$ to be forwarded to $\mathcal{F}_{Compute}$. When $\mathcal{F}_{ExtNet}$ (Algorithm 3) receives the message it removes the user's credentials and leaks the rest of the message to the adversary in Step 3. After receiving a (Confirm, session-id) message from the adversary in Step 4, $\mathcal{F}_{ExtNet}$ sends the original message on to $\mathcal{F}_{Compute}$. In Step 5, (Algorithm 4) $\mathcal{F}_{Compute}$ receives the message from the user and creates a request of the form ($\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{compute}$, $creds$, "Delete node-id ") to $\mathcal{F}_{Identity}$ via $\mathcal{F}_{SMT}$ in Step 6 (Algorithm 5). The message passing by $\mathcal{F}_{SMT}$ will proceed as in Steps 3-4. After $\mathcal{F}_{Identity}$ receives the validation message from $\mathcal{F}_{Compute}$ in Step 9 (Algorithm 6 )it will check that the user has permission to delete the requested node and send an appropriate response to $\mathcal{F}_{Compute}$, again via $\mathcal{F}_{SMT}$, in Step 10. Once $\mathcal{F}_{Compute}$ receives the validation message from $\mathcal{F}_{Identity}$ in Step 13, it will delete the requested node if the node exists and the validation was successful, otherwise it will note that the request failed. In Step 14, $\mathcal{F}_{Compute}$ will send a message to the user, proceeding as before through $\mathcal{F}_{ExtNet}$, notifying them of the result of their request — either the successful deletion of the node or its failure. The user, upon receiving this message in Step 17 will output the result to the environment in the final step.

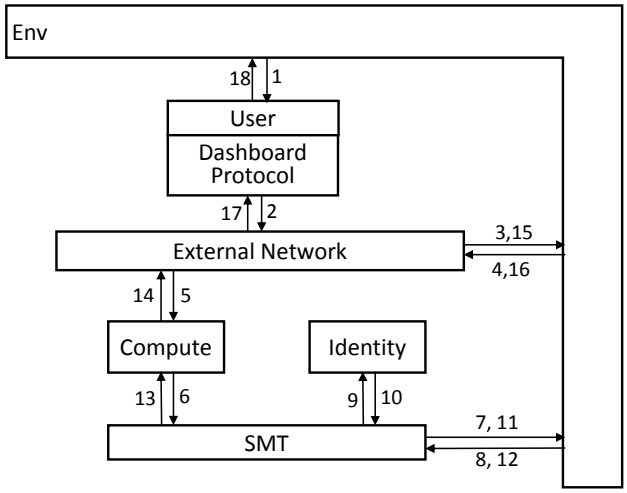


Figure 3: The Delete node functionality starts when the environment sends a "DeleteNode" message to the user. Next the user adds its credentials and sends it to the $\mathcal{F}_{Compute}$. Then, $\mathcal{F}_{Compute}$ creates a validation request to $\mathcal{F}_{Identity}$ in order to validate user's credentials. Based on the validation result and the node existence, $\mathcal{F}_{Compute}$ deletes the node and notifies the user. We describe the workflow in detail in Section 3.5.1.

### 3.5.2 Realization of This Workflow

We write specifications for our functionalities in pseudocode with relevant snippets for the DeleteNode workflow shown in Algorithms 2, 3, 4, 5, and 6.

**Algorithm 2** Dashboard Example Workflow (full version in Algorithm 9)

1: Upon receiving Request-Message from $E$:
2: User U Creates ($U_{Dashboard}$, $\mathcal{F}_{Service}$, Request-Message, *creds*);
3: Send-$\mathcal{F}_{ExtNet}$($U_{Dashboard}$, $\mathcal{F}_{Service}$, Request-Message, *creds*); ▷ Step 2
4:
5: Upon receiving (Sender, Output-Message) from $\mathcal{F}_{ExtNet}$: ▷ Step 17
6: Output (Output-Message) to $E$; ▷ Step 18

We allow the adversary to compromise users or services (Algorithms 7 shows corrupted Compute). A compromised user or service will reveal all of its internal state to the adversary and will from that point on be under full adversarial control. In particular, compromised users and services will not necessarily follow any specified protocol and can form their own messages at will. This is particularly relevant in our case because, upon compromising a service the adversary will learn not only its credentials, which are part of the internal state of the service, but also any credentials that service learns in the future. That is, any user that makes a request to a compromised service will also leak its own credentials to the adversary.

**Algorithm 3** External Network Example Workflow (full version in Algorithm 10)

1: Upon receiving (Sender, Receiver, Message): ▷ Step 2
2: Create New-Message by removing the credentials and replacing them by their IDs;
3: Send-Adversary(Sender, Receiver, New-Message); ▷ Step 3
4:
5: Upon receiving ("Confirm", session-id) from Adversary: ▷ Step 4
6: Send-Receiver(Sender, Message); ▷ Step 5

**Algorithm 4** Compute Example Workflow (full version in Algorithm 13)

1: Upon receiving (Receiver, "Delete Node", session-id, node-id, *creds*) from $\mathcal{F}_{ExtNet}$: ▷ Step 5
2: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{compute}$, *creds*, "Delete node-id "); ▷ Step 6
3:
4: Upon receiving ($\mathcal{F}_{Identity}$, "Service Validated", session-id, user-id, service-id, Request, valid) from $\mathcal{F}_{SMT}$: ▷ Step 13
5: **if** valid & a node with id=node-id exists **then**
6: Delete node with id=node-id;
7: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-id, Successful); ▷ Step 14
8: **else**
9: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-id, Fail); ▷ Alternate Step 14
10: **end if**

**Algorithm 5** SMT Example Workflow (full version in Algorithm 12)

1: Upon receiving *(Sender, Receiver,Message)*: ▷ Steps 6, 10
2: Create New-Message by removing the credentials and replacing them by their IDs;
3: Send-Adversary(Sender, Receiver, New-Message); ▷ Steps 7,11
4:
5: Upon receiving ("Confirm", session-id) from Adversary: ▷ Steps 8, 12
6: Send-Receiver(Sender, Message); ▷ Steps 13, 9

**Algorithm 6** Identity Example Workflow (full version in Algorithm 24)

1: Upon receiving (Receiver, "Service Validation", session-id, $creds_{service}$, *creds*, Request) from $\mathcal{F}_{SMT}$: ▷ Step 9
2: **if** $creds_{service}$ & *creds* are valid & *creds* is allowed to perform the Request **then**
3: valid=True;
4: **else**
5: valid=False;
6: **end if**
7: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Identity}$, *Receiver*, "Service Validated", session-id, user-id, service-id, Request, valid); ▷ Step 10

**Algorithm 7** Corrupted Compute

**function** ReceiveMessage
    Upon receiving (Source, message) from $\mathcal{F}_{SMT} \setminus \mathcal{F}_{ExtNet}$:
    Send-$\mathcal{F}_{SMT}$ (source, Adversary, message);
**end function**

**function** SendMessage
    Upon receiving (Adversary, "Forward", message, destination) from $\mathcal{F}_{SMT}$:
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, destination, message);
**end function**

**function** Main
    Upon receiving a message which does not contain "Forward" & Source=Adversary from $\mathcal{F}_{SMT}$:
    Apply the request;
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, Adversary, result of the request);
**end function**

# 4 Basic Security of OpenStack

The goal of this section is to analyze the security of OpenStack in the universal composability setting, specially, we want to prove the following theorem:

**Theorem 2.** *The OpenStack Services protocol from §3.5 UC-realizes the Ideal Cloud $\mathcal{F}_{Cloud}$ from §3.4 in the ($\mathcal{F}_{ExtNet}$, $\mathcal{F}_{SMT}$)-hybrid model.*

In order to prove this theorem, we need to:

- formally specifiy the ideal cloud and OpenStack Service in UC syntax
- provide a simulator and
- alalyze that the simulator works to insure that the views of the environment is the same in both worlds.

The remaining subsections of this section are exactly to cover these purposes. Here, we define the terminologies that have been used in the rest of this paper.

- session-id: A unique value which is concatenation of user identity U and an incremental number. The uniqueness of the session-id is evaluated in each functionality before sending to the request-message.
- Buffer: In order to be able to track a request, components needs to store the information related to that request. This is done by storing this information in Buffer and updating, finding or removing them later on.
- confirmation message: Represents the adversarial interruption. The adversary may:

– *Permit* the packet to be sent to the destination by sending a confirmation message for that session
– *Delay* the packet by sending the confirmation message with delay or
– *Drop* the packet by not sending the confirmation message at all.

## 4.1 Ideal Cloud Functionality Using Bearer Tokens

Algorithm 8 [3] represents $\mathcal{F}_{Cloud}$; the ideal cloud using bearer token in the case of service corruption. $\mathcal{F}_{Cloud}$ stores information related to corrupted services (Note that in this case, the services are only names for the corruption operation that is done by adversary) to be able to respond to the ideal-model adversary (i.e., to the simulator) in the case of corruption. Since the ideal cloud is a monolithic functionality, in the case on compromises it behave as if different components (services), such as Compute or Storage, have been corrupted despite not being composed of distinct parts. It does this by providing special interfaces for the simulator which allow it to request token validation or images if the environment has asked to corrupt some of the services. Additionally, if a portion of the cloud is corrupted it may cease to perform validation checks for its requests.

[3] The description of notation used in the algorithm is provided in appendix A

**Algorithm 8** Ideal Cloud with corrupted services for bearer token

**function** CLOUD
    Upon receiving ("Corrupt", Service, value) from Sim:
    Corrupt[Service]=value; % the default value=None. None indicates no corruption; Passive is the partial corruption, and Active is the full corruption
    **if** Service=$\mathcal{F}_{Image}$ & value!=None **then**
        **if** there are user requests that have not finalized **then**
            Add (user-id) to the List;
        **end if**
        send-Sim(DB of (image-id, corresponding-Image),List of (user-id));
    **else if** Service=$\mathcal{F}_{Compute}$ & value!=None **then**
        **if** there are user requests that have not finalized **then**
            Add (user-id) to the List;
            Send-Sim(List of (user-id));
        **end if**
    **else if** Service=$\mathcal{F}_{Identity}$ & value!=None **then**
        Send-Sim(DB of (user-id, roles));
    **else if** Service=$\mathcal{F}_{BlockStorage}$ & value!=None **then**
        **if** there are user requests that have not finalized **then**
            Add (user-id) to the List;
        **end if**
        Send-Sim(List of (user-id));
    **else if** Service=$\mathcal{F}_{Node}$node-id & value!=None **then**
        **if** there are user requests that have not finalized **then**
            Add (user-id) to the List;
        **end if**
        Send-Sim(List of (user-id));
    **else if** Service=$\mathcal{F}_{Volume}$volume-id & value!=None **then**
        **if** there are user requests that have not finalized **then**
            Add (user-id) to the List;
        **end if**
        Send-Sim(List of (user-id));
    **end if**
    Upon receiving (Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity) from user U:
    **if** session-id =(U,N) where N non-repetitive for user U **then**
        StoreBuffer (Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity);
    **end if**
    Send-Sim (Receiver, "Node Access", session-id, user-id, node-id, RequestActivity);

    Upon receiving("Result Access Node", session-id) from Sim:
    **if** FindBuffer(Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity, valid) **then**
        **if** valid=True **then**
            results ← EXECUTE(AccessRequest);

```
43:             UpdateBuffer({Receiver, "Access Node", session-id, creds, node-id, RequestActivity,
    valid}, {Receiver, "Access Node", session-id, creds, node-id, result, valid});
44:             Send-Sim ("Access Node", session-id, user-id, node-id, result, valid);
45:         end if
46:     end if
47:     Upon receiving("Output Access Node", session-id) from Sim:
48:     if FindBuffer(Receiver, "Access Node", session-id, creds, node-id, results, valid) then
49:         Valid=valid;
50:         RemoveBuffer("Access Node", session-id, creds, node-id, results, valid);
51:         Output("Access Node", session-id, results, Valid) to Receiver;
52:     end if
53:
54:     Upon receiving ("Access Node", session-id, node-id, RequestActivity) from Sim:
55:     if Corrupt[Node node-id ]=Active then
56:         results ← EXECUTE(AccessRequest);
57:     else
58:         results = None
59:     end if
60:     Send-Sim("Access Node", session-id, node-id, RequestActivity, results)
61:     Upon receiving (Receiver, "Attach Volume", session-id, creds, volume-id, node-id) from user
    U;
62:     if session-id =(U,N) where N non-repetitive for user U then
63:         StoreBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id);
64:     end if
65:     Send-Sim(Receiver, "Attach Volume", session-id, volume-id, node-id);
66:
67:     Upon receiving ("Result Attach Volume", session-id) from Sim:
68:     if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, valid) then
69:         if valid=True AND attached-volume for node node-id is None AND attached-node for
    volume volume-id is None then
70:             Set attached-volume for node node-id to volume-id
71:             Set attached-node for volume volume-id to node-id
72:             UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id,
    valid) }, {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success })
73:             Send-Sim("Attach Volume", session-id, volume-id, node-id, success)
74:         else
75:             UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id,
    valid) }, {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure) })
76:             Send-Sim("Attach Volume", session-id, volume-id, node-id, failure)
77:         end if
78:     end if
79:
```

```
80:     Upon receiving ("Output Attach Volume", session-id) from Sim:
81:     if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success) then
82:         RemoveBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success)
83:         Output("Attach Volume", session-id, success) to Receiver
84:     else if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure) then
85:         RemoveBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure)
86:         Output("Attach Volume", session-id, failure) to Receiver
87:     end if
88:
89:     Upon Receiving (Confirm, "Attach Volume" session-id, valid)
90:     if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id) then
91:         if user user-id is allowed to attach volume-id to node-id then
92:             valid=True
93:         else
94:             valid=False
95:         end if
96:         if Corrupt[Identity]=Active then
97:             valid = value sent by Sim
98:         end if
99:         UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id },
    {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, valid })
100:         Send-Sim(Receiver, "Attach Volume", session-id, volume-id, node-id, valid)
101:     end if
102:
103:     Upon receiving ("Attach Volume", session-id, volume-id, node-id) from Sim;
104:     if Corrupt[Compute]=Active OR Corrupt[Node node-id ]=Active then
105:         Set attached-volume for node node-id to volume-id
106:     end if
107:     if Corrupt[Storage]=Active OR Corrupt[Volume volume-id ]=Active then
108:         Set attached-node for volume volume-id to node-id
109:     end if
110:     Send-Sim ("Attach Volume", session-id, volume-id, node-id);
111:     Upon receiving (Receiver, "Detach Volume", session-id, creds, volume-id, node-id) from
    user U;
112:     if session-id =(U,N) where N non-repetitive for user U then
113:         StoreBuffer(Receiver, "Detach Volume", session-id, creds, volume-id, node-id);
114:     end if
115:     Send-Sim(Receiver, "Detach Volume", session-id, volume-id, node-id);
116:
117:     Upon receiving ("Result Detach Volume", session-id) from Sim:
118:     if FindBuffer(Receiver, "Detach Volume", session-id, creds, volume-id, node-id, valid) then
```

119: **if** valid=True AND attached-volume for node node-id is volume-id AND attached-node for volume volume-id is node-id **then**
120: Set attached-volume for node node-id to None
121: Set attached-node for volume volume-id to None
122: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid) }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success })
123: Send-Sim("Detach Volume", session-id, volume-id, node-id, success)
124: **else**
125: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid) }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure) })
126: Send-Sim("Detach Volume", session-id, volume-id, node-id, failure)
127: **end if**
128: **end if**
129:
130: Upon receiving ("Output Detach Volume", session-id) from Sim:
131: **if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success) **then**
132: RemoveBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success)
133: Output("Detach Volume", session-id, success) to Receiver
134: **else if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure) **then**
135: RemoveBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure)
136: Output("Detach Volume", session-id, failure) to Receiver
137: **end if**
138: Upon Receiving (Confirm, "Detach Volume" session-id, valid)
139: **if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id) **then**
140: **if** user user-id is allowed to detach volume-id from node-id **then**
141: valid=True
142: **else**
143: valid=False
144: **end if**
145: **if** Corrupt[Identity]=Active **then**
146: valid = value sent by Sim
147: **end if**
148: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid })
149: Send-Sim(Receiver, "Detach Volume", session-id, volume-id, node-id, valid)
150: **end if**
151:
152: Upon receiving ("Detach Volume", session-id, volume-id, node-id) from Sim;

153: **if** Corrupt[Compute]=Active OR Corrupt[Node node-id ]=Active **then**
154: Set attached-volume for node node-id to None
155: **end if**
156: **if** Corrupt[Storage]=Active OR Corrupt[Volume volume-id ]=Active **then**
157: Set attached-node for volume volume-id to None
158: **end if**
159: Send-Sim ("Detach Volume", session-id, volume-id, node-id);
160: $C_1-$ Upon receiving (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) from U:
161: **if** Corrupt[Compute]!="Active" **then**
162: **if** session-id =(U,N) where N non-repetitive for user U **then**
163: StoreBuffer (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure);
164: **end if**
165: **end if**
166: Send-Sim(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure);
167:
168: $C_3I_{12}-$ Upon receiving ("Create Node Output", session-id) from Sim:
169: **if** FindBuffer(Receiver, "Create Node Continue", session-id, node-id, valid) **then**
170: Output("Node Created", session-id, node-id, valid) to *Receiver*;
171: **end if**
172:
173: $D_1-$ Upon receiving (Receiver, "Delete Node", session-id, node-id, user-id) from user U:
174: **if** Corrupt[Compute]!="Active" **then**
175: **if** session-id =(U,N) where N non-repetitive for user U **then**
176: StoreBuffer (Receiver, "Delete Node", session-id, user-id, node-id);
177: **end if**
178: **end if**
179: Send-Sim(Receiver, "Delete Node", session-id, user-id, node-id);
180:
181: Upon receiving ("Output Delete Node", session-id) from Sim:
182: **if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id, Valid, NodeExist) **then**
183: **if** NodeExist=0 or Valid=0 **then**
184: Output("Delete Node", session-id, node-id, Fail) to *Receiver*;
185: **else**
186: Output("Delete Node", session-id, node-id, Successful) to *Receiver*;
187: **end if**
188: **end if**

189: Upon receiving ("Confirm", session-id) from Sim:
190: **if** FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) **then**
191: **if** user-id is valid & user-id is allowed to create node using Node-Structure **then**
192: **if** Corrupt[Image]="Active" **then**
193: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure})
194: $C_3I_1-$ Send-Sim (Receiver, "Create Node", session-id, user-id, "Corrupted Get Image");
195: **else**
196: **if** user-id is allowed to create node with image-id with Node-Structure & there exists image-id **then**
197: Corresponding-Image=get the image corresponding to image-id;
198: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image});
199: $C_4I-$ Send-Sim(Receiver, "Get Node ID", session-id, user-id, image-id, Corresponding-Image, Node-Structure);% GOTO $C_5I$
200: **else**
201: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node Continue", session-id, NULL, Fail});
202: $C_4II-$ Send-Sim(Receiver, "Create Node", session-id, user-id, Fail, reason="Get Image", Node-Structure); % GOTO $C_5II$
203: **end if**
204: **end if**
205: **else**
206: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node Continue", session-id, NULL, Fail});
207: $C_4III-$ Send-Sim(Receiver, "Create Node", session-id, user-id, Fail, reason="Create Node", Node-Structure); % GOTO $C_5III$
208: **end if**
209: **else if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id) **then**
210: **if** user-id is valid & user-id is allowed to delete node node-id **then**
211: Valid=1;
212: **else**
213: Valid=0;
214: **end if**
215: UpdateBuffer ({Receiver, "Delete Node", session-id, user-id, node-id },{Receiver, "Delete Node", session-id, user-id, node-id, Valid});
216: $D_4-$ Send-Sim (Receiver, "Delete Node", session-id, user-id, node-id, Valid);
217: **end if**
218:
219:

Upon receiving ("Service Validation", session-id, user-id, Request) from Sim:
**if** Corrupt[Compute]="Active" or Corrupt[Image]="Active" OR Corrupt[Storage]=Active **then**
    **if** user-id is valid & user-id is allowed to apply Request **then**
        valid=1;
    **else**
        valid=0;
    **end if**
    Send-Sim("validation", user-id, session-id, Request, valid);
**end if**

$D_3O_2-$ Upon receiving (Receiver, "Delete Node", session-id, node-id, valid) from Sim:
**if** Corrupt[Compute]="Active" or Corrupt[Identity]="Active" **then**
    Output("Delete Node", session-id, node-id, valid) to *Receiver*;
**end if**
$D_3ID_4-$ Upon receiving ("Delete Node", session-id, node-id, valid) from Sim:
**if** Flage-Identity="Active" & FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id) **then**
    **if** valid=1 **then**
        **if** node node-id exists **then**
            NodeExist=1;
            Delete node with node-id;
        **else**
            NodeExist=0
        **end if**
        RemoveBuffer(Receiver, "Delete Node", session-id, user-id, node-id)
        Send-Sim (Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist);
    **end if**
**end if**
$AC_2-$ Upon receiving ("MainComputeCorrupt", request) from Sim:
**if** Corrupt[Compute]="Active" **then**
    Apply the request;
    Send-Sim (MainComputeCorruptResult, result);
**end if**

$C_3C_1O_2-$ Upon receiving (Receiver, "Create Node result", session-id, node-id, valid) from Sim:
**if** Corrupt[Compute]="Active" **then**
    Output ("Node Created", session-id, node-id, valid) to *Receiver*;
**end if**

$C_3I_9-$ Upon receiving (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image, node-id) from Sim:
**if** Corrupt[Image]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure) **then**

261:
262: **if** node-id is not used **then**
263: UpdateBuffer({Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure}, {Receiver, "Create Node Continue", session-id, node-id, Successful});
264: Add (node-id, user-id, Corresponding-Image, Node-Structure) to the list of active nodes;
265: Send-Sim("Create Node Continue", session-id, user-id, successful);
266: **else**
267: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure,, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Fail});
268: Send-Sim("Create Node Continue", session-id, user-id, Fail); %GOTO $C_3I_{10}$
269: **end if**
270: **end if**
271:
272: $C_3C_1I_2-$ Upon receiving ("Get Image", session-id, user-id, image-id) from Sim:
273: **if** Corrupt[Compute]="Active" **then**
274: **if** user-id is valid & user-id is allowed to apply Request **then**
275: valid=1;
276: **if** image image-id exists **then**
277: Corresponding-Image= extract corresponding image of image-id;
278: **else**
279: Corresponding-Image=NULL;
280: **end if**
281: **else**
282: valid=0;
283: Corresponding-Image=NULL;
284: **end if**
285: Send-Sim("Get Image", session-id, user-id, image-id, Corresponding-Image, valid);%GOTO $C_3C_1I_3$
286: **end if**
287:
288: $C_3ID_8-$ Upon receiving ("Get Image", session-id, image-id) from Sim:
289: **if** Corrupt[Identity]="Active" **then**
290: **if** image image-id exists **then**
291: Corresponding-Image= extract corresponding image of image-id;
292: **else**
293: Corresponding-Image=NULL;
294: **end if**
295: Send-Sim("Get Image", session-id, image-id, Corresponding-Image);
296: **end if**
297:
298: $C_3ID_{12}-$ Upon receiving ("Node ID", session-id, node-id, Corresponding-Image) from Sim:
299: **if** Corrupt[Identity]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) **then**
300:

301:
302: **if** node-id is not used **then**
303: Add( node-id, user-id, Node-Structure, Corresponding-Image) to the list of active nodes;
304: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure}, {Receiver, "Create Node Continue, session-id, node-id, Successful});
305: Send-Sim("Create Node Continue", session-id, user-id, Successful);%GOTO $C_3I_{10}$
306: **else**
307: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure}, {Receiver, "Create Node Continue, session-id, node-id, Fail});
308: Send-Sim("Create Node Continue", session-id, user-id, Fail);%GOTO $C_3I_{10}$
309: **end if**
310: **end if**
311:
312: $C_3ID_5^*-$ Upon receiving ("Create Node Output valid", session-id, valid) from Sim:
313: **if** Corrupt[Identity]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) **then**
314: **if** valid=0 **then**
315: Output("Create Node", session-id, NULL, Fail) to *Receiver*;
316: **end if**
317: **end if**
318: $D_8-$ Upon receiving ("Delete Node", session-id, Continue) from Sim:
319: **if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id, Valid) **then**
320: **if** there is node with node-id **then**
321: NodeExist=1;
322: **else**
323: NodeExist=0;
324: **end if**
325: UpdateBuffer({Receiver, "Delete Node", session-id, user-id, node-id, Valid}, {Receiver, "Delete Node", session-id, user-id, node-id, Valid, NodeExist});
326: **if** Valid=1 & NodeExist=1 **then**
327: Delete node-id from the Node list;
328: **end if**
329: Send-Sim ("Delete Node Completed", session-id, Valid, NodeExist);
330: **end if**
331:
332: $D_{11}-$ Upon receiving ("Output Delete Node", session-id) from Sim:
333: **if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id, Valid, NodeExist) **then**
334: **if** NodeExist=0 or Valid=0 **then**
335: Output("Delete Node", session-id, node-id, Fail) to *Receiver*;
336: **else**
337: Output("Delete Node", session-id, node-id, Successful) to *Receiver*;
338: **end if**
339: **end if**
340:

341:

342: $C_{13}I-$ Upon receiving ("Node ID", session-id, node-id) from Sim:

343: **if** FindBuffer(Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image) **then**

344: node-struct=Node-Structure;

345: **if** node-id is not used **then**

346: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Successful});

347: Add (node-id, user-id, Corresponding-Image, Node-Structure) to the list of active nodes;

348: Send-Sim("Create Node Continue", session-id, user-id, successful);

349: **else**

350: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure,, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Fail});

351: Send-Sim("Create Node Continue", session-id, user-id, Fail);

352: **end if**

353: **end if**

354:

355: $C_9III, C_{13}II, C_{16}I-$ Upon receiving ("Create Node Output", session-id) from Sim:

356: **if** FindBuffer(Receiver, "Create Node Continue", session-id, node-id, valid) **then**

357: Output("Create Node", session-id, node-id, valid) to *Receiver*;

358: **end if**

359: **end function**

### 4.2 OpenStack Services Functionalities Using Bearer Token

In this section different OpenStack Service functionalities are explained.

For full-corrupt (active adversary), the code needs to be replaced with the *corrupt-code*, this is done by "Corrupt" function. In this case, the actual code is replaced by the corrupt-code which has three main functions:

- Receive Message: whenever the service receives a message it will forward it to the adversary and gives the control to the adversary.
- Send Message: whenever the service receives a message from adversary which is meant to be forwarded to another service, it sends the message to the destination service on behalf of the corrupted service.
- Main: whenever the service receives a message from the adversary which is not a forwarding message, it will apply the request as the adversary asks to.

**Dashboard**: It's the interface between the user/Environment and services (Algorithm 9). It gets the input from user/Environment changes it into format that is understandable by services and adds user credentials and sends it to the requested service and vices versa.

**Algorithm 9** Dashboard

1: **function** DASHBOARD
2: Upon receiving Request-Message from Env:
3: User U Creates ($U_{Dashboard}$, $F_{Service}$, Request-Message, *creds*);
4: Send-$\mathcal{F}_{ExtNet}$($U_{Dashboard}$, $F_{Service}$, Request-Message, *creds*);
5:
6: Upon receiving (Sender, Output-Message) from $\mathcal{F}_{ExtNet}$:
7: Output (Output-Message) to Env;
8: **end function**

**External Network**: Algorithm 10 represents the external network between the user/Environment and the cloud. In order to preserve confidentiality, whenever a message is received, the external network removes user credentials from the message and sends the shortened message to the adversary and waits for adversary. The request message will be forwarded to the destination whenever the adversary replies by confirmation message.

**Dummy Adversary** $\mathcal{A}$: We consider a dummy adversary, Algorithm 11, which by receiving any message, she will only forward it to the environment $E$ and waits for the environments response. The environment may either send a confirmation message, arbitrary message or a message that informs a corruption to a service. Next the $\mathcal{A}$ forwards the received message from $E$ to destination which is either the external network or SMT.

**Secure Message Transmission**: SMT (Algorithm 12) is the algorithm that represents the internal message queue of the cloud. It handles the communication between services by protecting integrity and confidentiality. Services communicate with each other through SMT, thus, any message that is sent from one service to another is transmitted through SMT. In order to handle the adversarial interruption, SMT removes credentials from the original message and gives the control to $\mathcal{A}$ by sending the shorten message. However, in the case of full-corruption (active adversary), the SMT will not remove the credential of the messages that are sent to the corrupted service.

**Algorithm 10** External Network

**function** ExternalNetwork
    Upon receiving (Sender, Receiver, Message):
    StoreBuffer(Sender, Receiver, Message);
    Create New-Message by removing the credentials and replacing them by their IDs; % i.e. Service-id, user-id
    Send-Adversary(Sender, Receiver, New-Message);

    Upon receiving ("Confirm", session-id) from Adversary:
    **if** FindBuffer (Sender, Receiver, Message) where it's session ID=session-id **then**
        RemoveBuffer(Sender, Receiver, Message) where it's session-ID=session-id;
        Send-Receiver(Sender, Message);
    **end if**
**end function**

**Algorithm 11** Dummy Adversary

**function** Adversary
    Upon receiving (message) from $\mathcal{F}_{ExtNet}/\mathcal{F}_{SMT}$:
    Send-Env(message);

    Upon receiving (message) from Env:
    Send-$\mathcal{F}_{ExtNet}/\mathcal{F}_{SMT}$ (message);

    Upon receiving ("Corrupt", Service, value, Corrupt-Code) from Env:
    Send-$\mathcal{F}_{SMT}$ ("Corrupt", Service, value, Corrupt-Code);
**end function**

**Algorithm 12** Secure Message Transmission

```
1: function SMT
2:     Upon receiving (Sender, Receiver,Message):
3:     if Sender=Adversary or Receiver=Adversary then
4:         Send-Receiver (Sender, Message);
5:     else
6:         StoreBuffer(Sender, Receiver, Message);
7:         if Flag-Sender=1 then
8:             Send-Adversary(Sender, Receiver, Message);
9:         else
10:            Create New-Message by removing the credentials and replacing them by their IDs;
11:            Send-Adversary(Sender, Receiver, New-Message);
12:        end if
13:    end if
14:
15:    Upon receiving ("Confirm", session-id) from Adversary:
16:    if FindBuffer (Sender, Receiver, Message) where it's session ID=session-id then
17:        RemoveBuffer(Sender, Receiver, Message);
18:        Send-Receiver(Sender, Message);
19:    end if
20:
21:    Upon receiving ("Corrupt", Service, value, Corrupt-Code) from Adversary:
22:    if value="Active" then
23:        Send-Service("Corrupt",Corrupt-Code);
24:    else
25:        Flag-Service=1;
26:    end if
27:
28: end function
```

**Compute**: Algorithm 13 represents compute service. This algorithm only covers "create node", "delete node", "access node", and "attach/detach volume" functionalities.

*Create Node function:* For a user to create a node, compute first checks with identity to determine whether the user is permitted to perform a create a node with the given structure and, if so, forwards the request to Image service. The image service will also check with identity to determine whether the user is permitted to access the image and, if so, returns the corresponding image back to compute. Based on the Image service result, the compute service may create the node and notify tge user with the result.

*Delete Node function:* For a user to delete a node, compute first checks with identity to determine whether the user is permitted to delete the node. If the user is allowed to delete the node and the node exists, compute deletes the node and notifies the user.

*Access Node function:* For a user to access a node compute first checks whether the user making the request is allowed to access the node and, if so, forwards the request to the intended node. The node will also check if the user issuing the request is allowed to make node accesses and will only execute the request if the user is permitted to do so. After completing or rejecting the request the node will return its response to compute which will forward it to the user who issued the request.

*Attach/Detach function:*In order to attach or detach a volume to/from a compute node a user issues a request to compute. Compute will check with identity to determine whether the user is permitted to perform attach/detach operations on that node and will request that the storage service make the volume available if so. The storage service will also validate the user's credentials and will only make the volume available if the user is permitted to attach/detach it. The storage service will then respond to compute and, if the volume is available, compute will indicate to the node that it should be attached/detached, and report the result to the user.

**Corrupted Compute**: Algorithm 14 represents compute service in full-corruption (active adversary). In this case, the actual code is replaced by the corrupt-code.

**Image**: Algorithm 15 represents only "get image" functionality of image service. Get Image functionality needs to access identity services for validating the the user and its request to get the image related to the image-id mentioned in the request-message.

**Corrupted Image**: Algorithm 16 represents image service in full-corruption (Active adversary).

**BlockStorage:** Algorithm 17 represents block storage service. This algorithm only covers "Attach/Detach volume" functionalities.

*Attach/Detach Volume function:* When compute service makes an attach/detach volume request, block storage first checks with identity to determine whether the user is permitted to perform attach/detach operations on that node. If the user is eligible and the volume-id exists, block storage makes a request to volume service to attach/detach the volume. The storage service will then respond to compute (if the volume is available).

**Corrupted BlockStorage:** Algorithm 18 represents block storage service in full-corruption (active adversary). In this case, the actual code is replaced by the corrupt-code.

**Volume:** Algorithm 19 represents Volume service. This algorithm only covers "Attach/Detach" and "access" functionalities.

*Attach/Detach function:* By receiving a request from block storage service, volume service makes sure that the volume-id is correct, then attaches/detaches the volume to/from the compute node.

*Access Node function:* This functionality performs the user's requested operation for a compute

**Algorithm 13** Compute

**function** CORRUPT
    Upon receiving ("Corrupt", Corrupt-Code) from Adv:
    Replace code of $\mathcal{F}_{Compute}$ with code of Corrupted $\mathcal{F}_{Compute}$
**end function**

**function** (ATTACH/DETACH) VOLUME
    Upon receiving (Receiver, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id) from $\mathcal{F}_{ExtNet}$:
    StoreBuffer (Receiver, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id);
    Send-$\mathcal{F}_{SMT}$ ( $\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{compute}$, *creds*, "(Attach/Detach) Volume");

    Upon receiving ($\mathcal{F}_{Identity}$, "Service Validated", session-id, user-id, service-id, Request, valid) from $\mathcal{F}_{SMT}$:
    **if** FindBuffer(Receiver, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id) **then**
        **if** valid=True **then**
            UpdateBuffer({Receiver, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id }, {Receiver, "Storage (Attach/Detach) Volume", session-id, *creds*, volume-id, node-id });
            Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{BlockStorage}$, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id);
        **else**
            RemoveBuffer(Receiver, "(Attach/Detach) Volume", session-id, *creds*, volume-id, node-id);
            Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "(Attach/Detach) Volume", session-id, volume-id, node-id, Fail);
        **end if**
    **end if**

    Upon receiving ($\mathcal{F}_{BlockStorage}$, Result, session-id) from $\mathcal{F}_{SMT}$:
    **if** FindBuffer(Receiver, "Storage (Attach/Detach) Volume", session-id, *creds*, volume-id, node-id) **then**
        **if** Result is Fail **then**
            RemoveBuffer(Receiver, "Storage (Attach/Detach) Volume", session-id, *creds*, volume-id, node-id);
            Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "(Attach/Detach) Volume", session-id, volume-id, node-id, Fail);
        **else**
            UpdateBuffer({Receiver, "Storage (Attach/Detach) Volume", session-id, *creds*, volume-id, node-id }, {$\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$,"(Attach/Detach) Volume", session-id, volume-id, node-id });
            Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, volume-id, node-id);
        **end if**
    **end if**

32:

33: Upon receiving ($\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, Result) from $\mathcal{F}_{SMT}$:

34: **if** FindBuffer($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, volume-id, node-id) **then**

35: **if** Result is Fail **then**

36: RemoveBuffer($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, volume-id, node-id);

37: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "(Attach/Detach) Volume", session-id, volume-id, node-id, Fail);

38: **else**

39: RemoveBuffer($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, volume-id, node-id);

40: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "(Attach/Detach) Volume", session-id, volume-id, node-id), Success;

41: **end if**

42: **end if**

43: **end function**

44:

45: **function** AccessNode

46: Upon receiving (Receiver, "Access Node", session-id, node-id, RequestActivity, *creds*):

47: StoreBuffer(Receiver, "Node Access", session-id, node-id, user-id, RequestActivity) from $\mathcal{F}_{ExtNet}$;

48: **if** node-id is in list of active Nodes & user user-id is allowed to access node-id **then**

49: UpdateBuffer({Receiver, "Node Access", session-id, node-id, user-id, RequestActivity},{Receiver, "Execute Node Access", session-id, node-id, user-id });

50: Send-SMT($\mathcal{F}_{Compute}$, $\mathcal{F}_{Node}$, "Execute Access Node", session-id, user-id, node-id, RequestActivity);

51: **else**

52: RemoveBuffer(Receiver, "Node Access", session-id, node-id, user-id, RequestActivity);

53: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Access Node", session-id, NULL, Fail);

54: **end if**

55:

56: Upon receiving ($\mathcal{F}_{Node}$, "Node Accessed", session-id, results, valid) from $\mathcal{F}_{SMT}$:

57: **if** FindBuffer(Receiver, "Execute Node Access", session-id, node-id, user-id) **then**

58: RemoveBuffer(Receiver, "Execute Node Access", session-id, node-id, user-id);

59: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Access Node", session-id, result, valid);

60: **end if**

61: **end function**

62:

63: **function** CREATENODE

64: Upon receiving (Receiver, "Create Node",session-id, *creds*, image-id, Node-Structure) from $\mathcal{F}_{ExtNet}$:

65: StoreBuffer (Receiver, "Validate Create Node", session-id, *creds*, image-id, Node-Structure);

66: Send-$\mathcal{F}_{SMT}$ ( $\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{compute}$, *creds*, "Create Node");

67:

68: Upon receiving ($\mathcal{F}_{Identity}$, "Service Validated", session-id, user-id, service-id, Request, valid) from $\mathcal{F}_{SMT}$:

69: **if** FindBuffer(Receiver, "Validate Create Node", session-id, *creds*, image-id, Node-Structure, node-id) **then**

70: **if** valid=True **then**

71: UpdateBuffer({Receiver, "Validate Create Node", session-id, *creds*, image-id, Node-Structure, valid}, {Receiver, "Get Image Create Node", session-id, *creds*, image-id, Node-Structure});

72: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, *creds*, image-id);

73: **else**

74: RemoveBuffer(Receiver, "Validate Create Node", session-id, *creds*, image-id, Node-Structure, valid);

75: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, NULL, Fail);

76: **end if**

77: **end if**

78:

79: Upon receiving ($\mathcal{F}_{Image}$, "Image", session-id, image-id, Corresponding-Image ) from $\mathcal{F}_{SMT}$:

80: **if** FindBuffer(Receiver, "Get Image Create Node", session-id, *creds*, image-id, Node-Structure) **then**

81: **if** Corresponding-Image = NULL **then**

82: RemoveBuffer(Receiver, "Get Image Create Node", session-id, *creds*, image-id, Node-Structure);

83: Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created",session-id, NULL, Fail);

84: **else**

UpdateBuffer({Receiver, "Get Image Create Node", session-id, *creds*, image-id, Node-Structure},{Receiver, "Request Node ID", session-id, *creds*, image-id, Node-Structure})
Send-$\mathcal{F}_{SMT}$ ( $\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);
**end if**
**end if**

Upon receiving (Adversary, "Node ID", session-id, node-id) from $\mathcal{F}_{SMT}$:
**if** FindBuffer(Receiver, "Request Node ID", session-id, *creds*, image-id, Node-Structure) **then**
node-struct=Node-Structure;
RemoveBuffer(Receiver, "Request Node ID", session-id, *creds*, image-id, Node-Structure);
**if** node-id is not used **then**
Add (node-id, user-id, node-struct) to the list of active nodes;
Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, Successful);
**else**
Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, Fail);
**end if**
**end if**
**end function**


**function** DeleteNode
Upon receiving (Receiver, "Delete Node", session-id, node-id, *creds*) from $\mathcal{F}_{ExtNet}$:
StoreBuffer (Receiver, "Validate Delete Node", session-id, *creds*, node-id);
Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{compute}$, *creds*, "Delete node-id ");

Upon receiving ($\mathcal{F}_{Identity}$, "Service Validated", session-id, user-id, service-id, Request, valid) from $\mathcal{F}_{SMT}$:
**if** FindBuffer(Receiver, "Validate Delete Node", session-id, *creds*, node-id) **then**
node-ID=node-id;
RemoveBuffer(Receiver, "Validate Delete Node", session-id, *creds*, node-id);
**if** valid=True & there is node with node-ID **then**
Delete node with node-ID;
Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-ID, Successful);
**else**
Send-$\mathcal{F}_{ExtNet}$ ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-ID, Fail);
**end if**
**end if**
**end function**

**Algorithm 14** Corrupted Compute

**function** RECEIVEMESSAGE
    Upon receiving (Source, message) from $\mathcal{F}_{SMT} \setminus \mathcal{F}_{ExtNet}$:
    Send-$\mathcal{F}_{SMT}$ (source, Adversary, message);
**end function**

**function** SENDMESSAGE
    Upon receiving (Adversary, "Forward", message, destination) from $\mathcal{F}_{SMT}$:
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, destination, message);
**end function**

**function** MAIN
    Upon receiving a message which does not contain "Forward" & Source=Adversary from $\mathcal{F}_{SMT}$:
    Apply the request;
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, Adversary, result of the request);
**end function**

**Algorithm 15** Image

**function** CORRUPT
    Upon receiving ("Corrupt", Corrupt-Code) from Adv:
    Replace code of $\mathcal{F}_{Image}$ with code of Corrupted $\mathcal{F}_{Image}$
**end function**
**function** GETIMAGE
    Upon receiving ($\mathcal{F}_{Compute}$, "Get Image", session-id, *creds*, image-id) from $\mathcal{F}_{SMT}$:
    **if** session-id =(U,N) where N non-repetitive for user U **then**
        StoreBuffer($\mathcal{F}_{Compute}$, "Get Image", session-id, *creds*, image-id);
        Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, $creds_{Image}$, *creds*, "Get Image image-id ");
    **end if**

    Upon receiving ($\mathcal{F}_{Identity}$, "Service Validated", session-id, user-id, service-id, Request, valid) from $\mathcal{F}_{SMT}$:
    **if** FindBuffer($\mathcal{F}_{Compute}$, "Get Image", session-id, *creds*, image-id) **then**
        Image-ID=image-id;
        RemoveBuffer(Receiver, "Get Image", session-id, *creds*, image-id);
        **if** valid=False OR No such image-id **then**
            Corresponding-Image=NULL;
        **else**
            Corresponding-Image=extract the corresponding image for Image-ID;
        **end if**
        Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-id, Image-ID, Corresponding-Image );
    **end if**
**end function**

**Algorithm 16** Corrupted Image

1: **function** ReceiveMessage
2: Upon receiving (Source, message) from $\mathcal{F}_{SMT} \setminus \mathcal{F}_{ExtNet}$:
3: Send-$\mathcal{F}_{SMT}$ (source, Adversary , message);
4: **end function**
5:
6: **function** SendMessage
7: Upon receiving (Adversary, "Forward", message, destination) from $\mathcal{F}_{SMT}$:
8: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Image}$, destination, message);
9: **end function**
10:
11:
12: **function** Main
13: Upon receiving a message which does not contain "Forward" & Source=Adversary from $\mathcal{F}_{SMT}$:
14: Apply the request;
15: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, Adversary, result of the request);
16: **end function**

node.

**Corrupted Volume:** Algorithm 20 represents volume service in full-corruption (active adversary). In this case, the actual code is replaced by the corrupt-code.

*ExecuteRequest function:* When compute service forwards an execution-request on behalf of the user to the intended node, the node will check if the user is allowed to make node accesses and if so, executes the request. After completing or rejecting the request the node will return its response to compute.

*Attach/Detach function:* When the compute service indicate to the node that it should attach/detach a specific volume, the node attaches/detaches the volume and notifies the compute.

**Partially Corrupted Node:** Algorithm refalg:PartiallyCorruptIdealNode represents Node's partial corruption (passive adversary). In this situation, some information are leaked to the adversary.

**Fully Corrupted Node:** Algorithm 23 represents Node in full-corruption (active adversary). In this case, the actual code is replaced by the corrupt-code..

**Identity**: Algorithm 24 gets validation request and validates the request based on its stored information.

**Corrupted Identity**: Algorithm 25 represents identity service in full-corruption (active adversary). In this case, the actual code is replaced by the corrupt-code.

**Algorithm 17** BlockStorage

```
1: function CORRUPT
2:     Upon receiving ("Corrupt", Corrupt-Code) from Adv:
3:     Replace code of F_BlockStorage with code of Corrupted F_BlockStorage
4: end function
5: function ATTACHVOLUME
6:     Upon receiving (Source, "Attach Volume", session-id, creds, volume-id, node-id)
7:     if Source is F_Compute then
8:         if session-id =(U,N) where N non-repetitive for user U then
9:             StoreBuffer(Source, "Attach Volume", session-id, creds, volume-id, node-id);
10:            Send-F_SMT (F_BlockStorage, F_Identity, "Service Validation", session-id, creds_Block, creds,
    "Attach Volume volume-id to node node-id ");
11:        end if
12:        Upon receiving (F_Identity, "Service Validated", session-id, user-id, service-id, Request,
    valid) from F_SMT:
13:        if FindBuffer(Source, "Attach Volume", session-id, creds, volume-id, node-id) then
14:            RemoveBuffer(Source, "Attach Volume", session-id, creds, volume-id, node-id);
15:            if valid=False OR No such volume-id then
16:                Send-F_SMT (F_BlockStorage, Source,"Fail", session-id, volume-id, node-id);
17:            else
18:                StoreBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Request,
    valid)
19:                Send-F_SMT (F_BlockStorage, F_Volume, "Attach", session-id, volume-id, node-id);
20:                Upon receiving (F_Volume, Response, session-id, volume-id, node-id) from F_SMT:
21:                FindBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Request,
    valid);
22:                if Request not in Buffer then
23:                    Ignore message
24:                else
25:                    RemoveBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Re-
    quest, valid);
26:                    Send-F_SMT (F_BlockStorage, Source, Response, session-id, volume-id, node-id);
27:                end if
28:            end if
29:        end if
30:    else
31:        Drop message
32:    end if
33: end function
```

```
function DetachVolume
    Upon receiving (Source, "Detach Volume", session-id, creds, volume-id, node-id)
    if Source is F_Compute then
        if session-id =(U,N) where N non-repetitive for user U then
            StoreBuffer(Source, "Detach Volume", session-id, creds, volume-id, node-id);
            Send-F_SMT (F_BlockStorage, F_Identity, "Service Validation", session-id, creds_Block, creds,
"Detach Volume volume-id fron Node node-id ");
        end if
        Upon receiving (F_Identity, "Service Validated", session-id, user-id, service-id, Request,
valid) from F_SMT:
        if FindBuffer(Source, "Detach Volume", session-id, creds, volume-id) then
            RemoveBuffer(Source, "Get Volume", session-id, creds, volume-id);
            if valid=False OR No such volume-id then
                Send-F_SMT (F_BlockStorage, Source, "Fail", session-id, volume-id, node-id);
            else
                StoreBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Request,
valid)
                Send-F_SMT (F_BlockStorage, F_Volume, "Detach", session-id, volume-id, node-id);
                Upon receiving (F_Volume, Response, session-id, volume-id, node-id) from F_SMT:
                FindBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Request,
valid);
                if Request not in Buffer then
                    Ignore message
                else
                    RemoveBuffer(F_Identity, "Service Validated", session-id, user-id, service-id, Re-
quest, valid);
                    Send-F_SMT (F_BlockStorage, Source, Response, session-id, volume-id, node-id);
                end if
            end if
        end if
    else
        Drop message
    end if
end function
```

**Algorithm 18** Corrupted BlockStorage

```
function ReceiveMessage
    Upon receiving (Source, message) from F_SMT \F_ExtNet:
    Send-F_SMT (source, Adversary, message);
end function
function SendMessage
    Upon receiving (Adversary, "Forward", message, destination) from F_SMT:
    Send-F_SMT (F_BlockStorage, destination, message);
end function
```

**Algorithm 19** Volume

1: **function** CORRUPT
2: Upon receiving ("Corrupt", Corrupt-Code) from Adv:
3: Replace code of $\mathcal{F}_{Volume}$ with code of Corrupted $\mathcal{F}_{Volume}$
4: **end function**
5: **function** ATTACH
6: Upon receiving (Sender, "Attach", session-id, user-id, volume-id, node-id) from $\mathcal{F}_{SMT}$:
7: **if** Sender is $\mathcal{F}_{BlockStorage}$ and volume-id is correct **then**
8: Set AttachedNode=node-id;
9: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, $\mathcal{F}_{BlockStorage}$, "Success", session-id);
10: **else**
11: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, $\mathcal{F}_{BlockStorage}$, "Fail", session-id);
12: **end if**
13: **end function**
14: **function** DETACH
15: Upon receiving (Sender, "Detach", session-id, user-id, volume-id, node-id) from $\mathcal{F}_{SMT}$:
16: **if** Sender is $\mathcal{F}_{BlockStorage}$ and volume-id is correct and AttachedNode=node-id **then**
17: Set AttachedNode=None;
18: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, $\mathcal{F}_{BlockStorage}$, "Success", session-id);
19: **else**
20: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, $\mathcal{F}_{BlockStorage}$, "Fail", session-id);
21: **end if**
22: **end function**
23: **function** ACCESS
24: Upon receiving (Sender, "Access", operation, session-id, user-id, volume-id, node-id) from $\mathcal{F}_{SMT}$:
25: **if** Sender is AttachedNode and volume-id is correct **then**
26: perform operation
27: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, Sender, "Success", data requested, session-id);
28: **else**
29: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, Sender, "Fail", session-id);
30: **end if**
31: **end function**

**Algorithm 20** Corrupted Volume

1: **function** ReceiveMessage
2: Upon receiving (Source, message) from $\mathcal{F}_{SMT} \setminus \mathcal{F}_{ExtNet}$:
3: Send-$\mathcal{F}_{SMT}$ (source, Adversary, message);
4: **end function**
5: **function** SendMessage
6: Upon receiving (Adversary, "Forward", message, destination) from $\mathcal{F}_{SMT}$:
7: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Volume}$, destination, message);
8: **end function**
9: **function** Main
10: Upon receiving a message which does not contain "Forward" & Source=Adversary from $\mathcal{F}_{SMT}$:
11: Apply the request;
12: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, Adversary, result of the request);
13: **end function**

**Algorithm 21** Node

```
1: function CORRUPT
2:     Upon receiving ("Corrupt", Corrupt-Code) from Adv:
3:     if type='full' then
4:         Replace code of F_Node with code of Fully Corrupted F_Node
5:     else if type='partial' then
6:         Replace code of F_Node with code of Partially Corrupted F_Node
7:     end if
8: end function
9: function ATTACH
10:    Upon receiving (Sender, "Attach", volume-id) from F_SMT:
11:    if Source is F_Compute then
12:        if attached-volume is None then
13:            Set attached-volume to volume-id
14:            Send-F_SMT (F_Node, F_Compute, "Attach Volume", session-id, Success);
15:        else
16:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Fail);
17:        end if
18:    else
19:        drop message
20:    end if
21: end function
22:
23: function DETACH
24:    Upon receiving (Sender, "Detach", volume-id) from F_SMT:
25:    if Source is F_Compute then
26:        if attached-volume is volume-id then
27:            Set attached-volume to None
28:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Success);
29:        else
30:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Fail);
31:        end if
32:    else
33:        drop message
34:    end if
35: end function
```

36: **function** EXECUTEREQUEST
37: Upon receiving ($\mathcal{F}_{Compute}$, "Execute Access Node", session-id, user-id, node-id, AccessRequest) from $\mathcal{F}_{SMT}$:
38: **if** session-id =(U,N) where N non-repetitive for user U and U is the owner of the node **then**
39: **if** AccessRequest requires data **then**
40: StoreBuffer($\mathcal{F}_{Node}$, $\mathcal{F}_{Volume}$, "Access", data specified by AccessRequest, session-id, user-id, node-id)
41: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Node}$, $\mathcal{F}_{Volume}$, "Access", data specified by AccessRequest, session-id, user-id, volume-id, node-id)
42: Upon receiving ($\mathcal{F}_{Volume}$, $\mathcal{F}_{Node}$, "Success", data, session-id) from $\mathcal{F}_{SMT}$:
43: Store returned data
44: **end if**
45: results ← EXECUTE(AccessRequest);
46: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Node}$, $\mathcal{F}_{Compute}$, "Node Accessed", session-id, results, Successful);
47: **else**
48: Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Node}$, $\mathcal{F}_{Compute}$, "Node Accessed", session-id, None, Fail);
49: **end if**
50: **end function**

**Algorithm 22** Partially Corrupt Node

```
1: function CORRUPT
2:     Upon receiving ("Corrupt", Corrupt-Code) from Adv:
3:     if type='full' then
4:         Replace code of F_Node with code of Fully Corrupted F_Node
5:     end if
6: end function
7: function ATTACH
8:     Upon receiving (Sender, "Attach", volume-id) from F_SMT:
9:     if Source is F_Compute then
10:        if attached-volume is None then
11:            Set attached-volume to volume-id
12:            Send-F_SMT (F_Node, F_Compute, "Attach Volume", session-id, Success);
13:        else
14:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Fail);
15:        end if
16:    else
17:        drop message
18:    end if
19: end function
20:
21: function DETACH
22:    Upon receiving (Sender, "Detach", volume-id) from F_SMT:
23:    if Source is F_Compute then
24:        if attached-volume is volume-id then
25:            Set attached-volume to None
26:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Success);
27:        else
28:            Send-F_SMT (F_Node, F_Compute, "Detach Volume", session-id, Fail);
29:        end if
30:    else
31:        drop message
32:    end if
33: end function
```

```
34:
35: function ExecuteRequest
36:     Upon receiving (F_Compute, "Execute Access Node", session-id, user-id, node-id, AccessRequest) from F_SMT:
37:     if session-id =(U,N) where N non-repetitive for user U and U is the owner of the node then
38:         if AccessRequest requires data then
39:             StoreBuffer(F_Node, F_Volume, "Access", operation specified by AccessRequest, session-id, user-id, node-id)
40:             Send (F_Node, F_Volume, "Access", operation specified by AccessRequest, session-id, user-id, node-id)
41:             Upon receiving (F_Volume, F_Node, "Success", data, session-id) from F_SMT:
42:             Store returned data
43:         end if
44:         results ← Execute(AccessRequest);
45:         Send (Adversary, F_Node, F_Volume, "Access", operation specified by AccessRequest, session-id, user-id, node-id, results, successful)
46:         Send-F_SMT (F_Node, F_Compute, "Node Accessed", session-id, results, Successful);
47:     else
48:         Send (Adversary, F_Node, F_Volume, "Access", operation specified by AccessRequest, session-id, user-id, node-id, failure)
49:         Send-F_SMT (F_Node, F_Compute, "Node Accessed", session-id, None, Fail);
50:     end if
51: end function
```

**Algorithm 23** Fully Corrupt Node

```
1: function ReceiveMessage
2:     Upon receiving (Source, message) from F_SMT \F_ExtNet:
3:     Send-F_SMT (source, Adversary, message);
4: end function
5: function SendMessage
6:     Upon receiving (Adversary, "Forward", message, destination) from F_SMT:
7:     Send-F_SMT (F_Node, destination, message);
8: end function
9:
10: function Main
11:     Upon receiving a message which does not contain "Forward" & Source=Adversary from F_SMT:
12:     Apply the request;
13:     Send-F_SMT (F_Compute, Adversary, result of the request);
14: end function
```

**Algorithm 24** Identity

**function** CORRUPT
    Upon receiving ("Corrupt", Corrupt-Code) from Adv:
    Replace code of $\mathcal{F}_{Identity}$ with code of Corrupted $\mathcal{F}_{Identity}$
**end function**
**function** SERVICEVALIDATION
    Upon receiving (Receiver, "Service Validation", session-id, $creds_{service}$, *creds*, Request) from $\mathcal{F}_{SMT}$:
    **if** $creds_{service}$ & *creds* are valid & *creds* is allowed to perform the Request **then**
        valid=True;
    **else**
        valid=False;
    **end if**
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Identity}$, *Receiver*, "Service Validated", session-id, user-id, service-id, Request, valid);
**end function**

**Algorithm 25** Corrupt Identity

**function** RECEIVEMESSAGE
    Upon receiving (Source, message) from $\mathcal{F}_{SMT} \setminus \mathcal{F}_{ExtNet}$:
    Send-$\mathcal{F}_{SMT}$ (source, Adversary, message);
**end function**
**function** SENDMESSAGE
    Upon receiving (Adversary, "Forward", message, destination) from $\mathcal{F}_{SMT}$:
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Identity}$, destination, message);
**end function**

**function** MAIN
    Upon receiving a message which does not contain "Forward" & Source=Adversary from $\mathcal{F}_{SMT}$:
    Apply the request;
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Compute}$, Adversary, result of the request);
**end function**

### 4.3 Simulator for Ideal Cloud Using Bearer Tokens

We show the complete simulator $S$ for the ideal cloud using bearer tokens in Algorithm 26[4]. Here, the simulator gets input from ideal cloud and environment $E$ and based on the conditions it either response to or convert them into intermediate messages and sends them to the desired destination. $S$ buffers received information from the ideal cloud to make future decisions and keep the state of each request.

In the OpenStack Services world, whenever a service is corrupted, the adversary gets access to the information that are stored in the services database such as user credentials which have unfinalized request or have made a new request. This situation should also be correct in the ideal Cloud/Simulator world. Therefore, in the case of corruption, the cloud sends some extra information such as list of affected user-id to $S$. The simulator then creates and stores credential per each user-id and use it as needed in the following messages.

Note that, since the bearer token is an unscoped multi usage token, the adversary is able to make different requests on behalf of the compromised user. Also, she is able to send the same request more than once to another service. To realize these properties, the simulator only checks the credential and hands the request message to the cloud and responses based on the results it receives from the cloud.

### 4.4 Analysis

Here, we demonstrate that the ideal cloud UC-emulates the OpenStack Services. As with most UC analyses, our argument proceeds via induction on the steps taken by the environment $E$. We show that for any message that $E$ might send, the next incoming message received by $E$ maintains the invariant that *$E$'s view in the ideal cloud and OpenStack Services is identical*. More specifically, for any state that $E$ can reach, the action of the simulator $S$ up to that point must have ensured that $E$'s view is the same in both worlds. Ergo, $E$ cannot distinguish whether it is interacting with the OpenStack Services or with the composition of ideal cloud and the simulator.

To improve the presentation of this proof, we make the argument in stages. We begin with a 'warmup' analysis (property 1) that makes two simplifying assumptions: (i) $E$ permits each OpenStack command to complete before executing the next one and (ii) $E$ has not compromised any OpenStack services. We then remove those assumptions in pieces. The operation of the ideal cloud and OpenStack Services can change when $E$ is permitted to interleave commands, yet fortunately both clouds change in identical ways (property 2). Finally, permitting $E$ to compromise services causes yet more changes to the behavior, and does so in a manner that depends on how the token mechanism works, yet once again our invariant is maintained (property 3). Looking ahead, in Sections 5 and 6 we will introduce an improved 'one-time' token mechanism and analyze its security. As a result, we purposely provide our proof of property 3 in a manner that can be partially re-used when considering other token mechanisms.

In our attempt to balance comprehensiveness with comprehension, we explain each property for only one combination of the ideal cloud commands (attach/detach volume, access node, create node, or get image) and with a focus only on those OpenStack services most pertinent to the command. We believe that our choices are 'representative' of the properties we wish to demonstrate, in the sense that they showcase all of the insights involved in the analysis without subjecting the reader to redundant case analyses for the remaining commands and services.

[4]The description of notation used in the algorithm is provided in appendix A

**Algorithm 26** Simulator with corrupted services for bearer token

```
1: function SIMULATOR(message)
2:     Upon receiving ("Corrupt", Service, value, corrupt-code) from Env:
3:     Corrupt[Service]=value;
4:     if FindServiceCorrupt(Service, Service-id, creds_service) & creds_service =NULL then
5:         Credential= Generate credential for the service;
6:         creds_service =Credential;
7:         UpdateServiceCorrupt(Service, Service-id, creds_service);
8:     end if
9:     Send-IdealCloud("Corrupt", Service,value);
10:
11:    Upon receiving (DB of (image-id, corresponding-Image),List of (user-id)) from Ideal Cloud:
12:    StoreDB (All (image-id, corresponding-Image));
13:    Create unique Credential per each user-id in the List if it has not been created before;
14:    StoreBufferUserCorrupt(user-id,Credential);
15:
16:    Upon receiving DB of (user-id, roles))from Ideal Cloud:
17:    Create unique Credential per each user-id in the List if it has not been created before;
18:    StoreBufferUserCorrupt(user-id,Credential, roles);
19:
20:    Upon receiving List of (user-id)from Ideal Cloud:
21:    Create unique Credential per each user-id in the List if it has not been created before;
22:    storeBufferUserCorrupt(user-id,Credential);
23:
24:    Upon receiving (Sender, Receiver, message) from Env:
25:    if Receiver=F_Compute & Corrupt[Compute]="Active" then
26:        Send-Env(Sender, Adversary, message);
27:    else if Receiver=F_Identity & Corrupt[Identity]="Active" then
28:        Send-Env(Sender, Adversary, message);
29:    else if Receiver=F_Image & Corrupt[Image]="Active" then
30:        Send-Env(Sender, Adversary, message);
31:    else if Receiver=F_BlockStorage & Corrupt[Storage]="Active" then
32:        Send-Env(Sender, Adversary, message);
33:    else if Receiver=F_Node node-id & Corrupt[Node node-id ]="Active" then
34:        Send-Env(Sender, Adversary, message);
35:    else if Receiver=F_Volume volume-id & Corrupt[Volume volume-id ]="Active" then
36:        Send-Env(Sender, Adversary, message);
37:    end if
38:
```

39:

40: Upon receiving (Receiver, "Node Access", session-id, user-id, node-id, RequestActivity) from Cloud:

41: StoreBuffer(Receiver, "Node Access", session-id, user-id, node-id, RequestActivity)

42: Send-Env ($\mathcal{F}_{Compute}$, "Node Access", session-id, user-id, node-id, RequestActivity)

43:

44: Upon receiving ("Access Node", session-id, user-id, node-id, result, valid)

45: StoreBuffer("Access Node", session-id, user-id, node-id, result, valid)

46: Send-Env ($\mathcal{F}_{Compute}$, "Access Node", session-id, user-id, node-id, result, valid)

47:

48: Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Access Node", session-id, node-id, RequestActivity) from Env:

49: Send-IdealCloud(Adversary, "Access Node", session-id, node-id, RequestActivity)

50:

51: Upon Receiving ("Access Node", session-id, node-id, RequestActivity, results) from Ideal-Cloud

52: Send-Env("Access Node", session-id, node-id, RequestActivity, results);

53:

54: Upon receiving (Receiver, "Attach Volume", session-id, user-id, node-id) from Cloud:

55: StoreBuffer(Receiver, "Attach Volume", session-id, user-id, node-id)

56: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, user-id, node-id)

57:

58: Upon Receiving("Attach Volume", session-id, volume-id, node-id, success)

59: StoreBuffer("Attach Volume", session-id, volume-id, node-id, success)

60: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, volume-id, node-id, success)

61:

62: Upon Receiving("Attach Volume", session-id, volume-id, node-id, failure)

63: StoreBuffer("Attach Volume", session-id, volume-id, node-id, failure)

64: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, volume-id, node-id, failure)

65:

66: Upon Receiving("Attach Volume", session-id, volume-id, node-id, valid)

67: StoreBuffer("Attach Volume", session-id, volume-id, node-id, valid)

68: Send-Env ($\mathcal{F}_{Identity}$, "Attach Volume", session-id, volume-id, node-id, valid)

69:

70: Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Attach Volume", session-id, node-id, RequestActivity) from Env:

71: Send-IdealCloud(Adversary, "Attach Volume", session-id, node-id, RequestActivity);

72:

73:

74: Upon Receiving ("Attach Volume", session-id, volume-id, node-id)

75: **if** (Compute is Corrupt OR Node node-id is Corrupt) AND (Storage is Corrupt OR Volume volume-id is Corrupt) **then**

76: Send-Env("Attach Volume", session-id, volume-id, node-id, success)

77: **else**

78: Send-Env("Attach Volume", session-id, volume-id, node-id, failure)

79: **end if**

80:

81: Upon receiving (Receiver, "Detach Volume", session-id, user-id, node-id) from Cloud:

82: StoreBuffer(Receiver, "Detach Volume", session-id, user-id, node-id)

83: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, user-id, node-id)

84:

85: Upon Receiving("Detach Volume", session-id, volume-id, node-id, success)

86: StoreBuffer("Detach Volume", session-id, volume-id, node-id, success)

87: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, volume-id, node-id, success)

88:

89: Upon Receiving("Detach Volume", session-id, volume-id, node-id, failure)

90: StoreBuffer("Detach Volume", session-id, volume-id, node-id, failure)

91: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, volume-id, node-id, failure)

92:

93: Upon Receiving("Detach Volume", session-id, volume-id, node-id, valid)

94: StoreBuffer("Detach Volume", session-id, volume-id, node-id, valid)

95: Send-Env ($\mathcal{F}_{Identity}$, "Detach Volume", session-id, volume-id, node-id, valid)

96:

97: Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Detach Volume", session-id, node-id, RequestActivity) from Env:

98: Send-IdealCloud(Adversary, "Detach Volume", session-id, node-id, RequestActivity)

99:

100: Upon Receiving ("Detach Volume", session-id, volume-id, node-id)

101: **if** (Compute is Corrupt OR Node node-id is Corrupt) OR (Storage is Corrupt OR Volume volume-id is Corrupt) **then**

102: Send-Env("Detach Volume", session-id, volume-id, node-id, success)

103: **else**

104: Send-Env("Detach Volume", session-id, volume-id, node-id, failure)

105: **end if**

106:

107:

108: $C_2-$ Upon receiving (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) from Ideal Cloud:

109: StoreBuffer(Receiver,"Create Node", session-id, user-id, image-id, Node-Structure);

110: Send-Env(Receiver, $\mathcal{F}_{Compute}$, "Create Node", session-id, user-id, image-id, Node-Structure);

111:

112: Upon receiving ("validation",user-id, session-id, Request, valid) from IdealCloud:

113: **if** FindBuffer("Result, Service, destination, message) with request-ID=session-id **then**

114: UpdateBuffer({"Result", $F_{Service}$, destination, message}, {Receiver=destination, Sender= $F_{Service}$, "Service validated", session-id, Request, valid});

115: Send-Env(Receiver, Sender, "Service validated", Request, valid);

116: **end if**

117:

118: $AC_1-$ Upon receiving (Adversary, $\mathcal{F}_{Compute}$, request) from Env:

119: **if** Corrupt[Compute]="Active" **then**

120: Send-IdealCloud("MainComputeCorrupt", request);

121: **end if**

122:

123: $C_3C_1I_3-$ Upon receiving ("Get Image", session-id, Corresponding-Image, valid) from IdealCloud:

124: **if** FindBuffer("Result, Service, destination, message) with request-ID=session-id **then**

125: UpdateBuffer({"Result", $F_{Service}$, destination, message}, {$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid});

126: StoreDB(image-id, Corresponding-Image);% if this image is not stored in the Sim DB, then store it.

127: Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id FImage-id, user-id, "Get Image image-id ");

128: **end if**

129:

130: $C_3I_2-$Upon receiving (Receiver, "Create Node", session-id, user-id, "Corrupted Get Image") from Ideal Cloud:

131: StoreBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image");

132: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node");

133:

134: $C_3ID_9-$ Upon receiving ("Get Image", session-id, image-id, Corresponding-Image) from Ideal Cloud:

135: **if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid) **then**

136: **if** Corresponding-Image=NULL **then**

137: valid=0;

138: **else**

139: valid=1;

140: **end if**

141:

UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid},{Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid});

Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-id, image-id, , Corresponding-Image);

**end if**

$C_3I_{10}$−Upon receiving ("Create Node Continue", session-id, user-id, valid) from Ideal-Cloud:

**if** FindBuffer(Receiver, "Node ID continue", session-id, node-id) **then**

UpdateBuffer({Receiver, "Node ID continue", session-id },{Receiver, "Create Node Output", session-id })

Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);%GOTO $C_3I_{11}$

**else if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id) **then**

UpdateBuffer ({Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id },{Receiver, "Create Node Output", session-id });

Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);%GOTO $C_3I_{11}$

**end if**

$C_{12}I, C_3I_8, C_3ID_{11}$− Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Node ID", session-id, node-id) from Env:

**if** Corrupt[Image]="Active" & FindBuffer (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image) **then**

UpdateBuffer({Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image},{Receiver, "Node ID continue", session-id, node-id });

$C_3ID_8$− Send-IdealCloud (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image, node-id);

**else if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid) **then**

UpdateBuffer({Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid},{Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id })

$C_3ID_{11}$−Send-IdealCloud("Node ID", session-id, node-id, Corresponding-Image);

**else if** FindBuffer(Receiver, "Request Node ID", session-id) **then**

UpdateBuffer ({Receiver, "Request Node ID", session-id },{Receiver, "Node ID continue", session-id, node-id })

Send-IdealCloud ("Node ID", session-id, node-id);

**end if**

167:

168: Upon receiving (Adversary, Receiver, "Forward", message, destination) from Env:

169: **if** Receiver=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" **then**

170: **if** Destination is not a service **then**

171: StoreBuffer("Output", $\mathcal{F}_{Compute}$, destination, message);

172: **else**

173: StoreBuffer($\mathcal{F}_{Compute}$, destination, message);

174: **end if**

175: $C_3C_1-$ Send-Env($\mathcal{F}_{Compute}$, destination, message);% TO FOLLOW $C_3C_1$ (OUTPUT GOTO $C_3C_1O_1$, ELSE GOTO $C_3C_1ID_1$ [identity] OR $C_3C_1I_1$ [glance])

176: **else if** Receiver=$\mathcal{F}_{Identity}$ & Corrupt[Identity]="Active" **then**

177: **if** destination=$\mathcal{F}_{Compute}$ & Request="Delete Node" & message is "Service Validated" type & FindBuffer(Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id) **then**

178: extract the valid value from the message;

179: UpdateBuffer({Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid});

180: $D_3ID_2-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);

181: **else if** destination=$\mathcal{F}_{Compute}$ & Request="Create Node with Node-Structure" & message is "Service Validated" type & FindBuffer(Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure) **then**

182: extract the valid value from the message;

183: UpdateBuffer({Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure}, {Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid});

184: $C_3ID_2-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);

185: **else if** destination=$\mathcal{F}_{Image}$ & Request="Get Image image-id " & message is "Service Validated" type & FindBuffer(Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid) **then**

186: extract the Image-valid value from the message;

187: UpdateBuffer({Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid}, {Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid});

188: $C_3ID_6-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);

189: **else**

190: StoreBuffer($\mathcal{F}_{Identity}$, destination, message);

191: Send-Env($\mathcal{F}_{Identity}$, destination, message);

192: **end if**

193: **else if** Receiver=$\mathcal{F}_{Image}$ & Corrupt[Image]="Active" **then**

194: **if** destination= $\mathcal{F}_{Compute}$ & message=("Image", session-id, image-id, Corresponding-Image ) & FindBuffer(Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image") where the message session-ID=session-id **then**

195: UpdateBuffer ({Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image});

196: **else**

197: StoreBuffer($\mathcal{F}_{Image}$, destination, message);

198: **end if**

199: $C_3I_6-$ Send-Env($\mathcal{F}_{Image}$, destination, message);%GOTO $C_3I_7ID_1$ [IDENTITY] OR $C_3I_7$ [COMPUTE]

200: **end if**

201:

202: $C_3I_6^*-$ Upon receiving (Adversary, $\mathcal{F}_{Image}$, "Get Image", image-id) from Env:

203: **if** Corrupt[Image]="Active" **then**

204: **if** image-id exist **then**

205: Corresponding-Image= extract the corresponding image for image-id;

206: **else**

207: Corresponding-Image= NULL;

208: **end if**

209: Send-Env ($\mathcal{F}_{Image}$, Adversary, "Image", image-id, Corresponding-Image );

210: **end if**

211:

212: $D_2-$ Upon receiving (Receiver, "Delete Node", session-id, user-id, node-id) from Ideal cloud:

213: StoreBuffer(Receiver,"Delete Node", session-id, user-id, node-id);

214: Send-Env(Receiver, $\mathcal{F}_{Compute}$, "Delete Node", session-id, node-id, user-id);

215:

216: $AC_3-$ Upon receiving ("MainComputeCorruptResult", result) from IdealCloud:

217: Send-Env ($\mathcal{F}_{Compute}$, Adversary, result); % GOTO $C_3C_1$ (for forwarding), GOTO $AC_1$ (for Adv to make new request)

218:

219: $D_3ID_5-$ Upon receiving (Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist) from Ideal Cloud:

220: StoreBuffer(Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist);

221: **if** valid=1 & NodeExist=1 **then**

222: Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, successful );

223: **else**

224: Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, Fail );

225: **end if**

226:

227:

228: Upon receiving ("Confirm", session-id) from Env:

229: **if** FindBuffer(Receiver,"Create Node", session-id, user-id, image-id, Node-Structure) **then**

230: **if** Corrupt[Compute]="Active" **then**

231: **if** FindBufferUserCorrupt(user-id, Credential) **then**

232: *creds* =Credential;

233: **else**

234: Credential= Generate credential for the user;

235: *creds* = Credential;

236: StoreBufferUserCorrupt(user-id,Credential);

237: **end if**

238: $C_3C-$ Send-Env(Receiver, Adversary, "Create Node", session-id, *creds*, image-id, Node-Structure);% GOTO $AC_1$ (Adv makes a new request), $C_3C_1$ (for forwarding)

239: **else if** Corrupt[Identity]="Active" **then**

240: UpdateBuffer({Receiver,"Create Node", session-id, user-id, image-id, Node-Structure},{Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure});

241: Request="Create Node with Node-Structure";

242: $C_3ID-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Service-id, user-id, Request);

243: **else if** Corrupt[Image]="Active" **then**

244: $C_3I-$ Send-IdealCloud("Confirm", session-id);

245: **else**

246: $C_3-$ Send-IdealCloud("Confirm", session-id);

247: **end if**

248: **else if** FindBuffer(Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure) **then**

249: **if** Corrupt[Identity]="Active" **then**

250: **if** FindBufferUserCorrupt(user-id, Credential,roles) **then**

251: *creds* =Credential;

252: **else**

253: Credential= Generate credential for the user;

254: *creds* = Credential;

255: StoreBufferUserCorrupt(user-id,Credential,roles);

256: **end if**

257: Extract $creds_{service}$ from ServiceCorrupt(Service, Service-id, $creds_{service}$) where Service=$\mathcal{F}_{Compute}$;

258: UpdateBuffer({Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure},{Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure});

259: $C_3ID_1-$ Send-Env ($\mathcal{F}_{Compute}$, Adversary, "Service validation, session-id, $creds_{service}$, *creds*, Request); % GOTO $D_3ID_2$ OR ADV MAY SEND ARBITRARY MESSAGE TO A SERVICE

260: **end if**

261: **else if** FindBuffer(Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid) **then**

262: **if** Corrupt[Identity]="Active" **then**
263: **if** valid=0 **then**
264: UpdateBuffer({Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, valid});
265: $C_3ID_3^*-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$
266: **else**
267: UpdateBuffer({Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid});
268: $C_3ID_3-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id);%GOTO $C_3ID_4$
269: **end if**
270: **end if**
271: **else if** FindBuffer(Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, valid) **then**
272: $C_3ID_4^*-$ Send-IdealCloud("Create Node Output valid", session-id, Fail);
273: **else if** FindBuffer(Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid) **then**
274: UpdateBuffer({Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid});
275: $C_3ID_4-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, compute-id, user-id, Request="Get Image image-id ");
276: **else if** FindBuffer(Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid) **then**
277: **if** Corrupt[Identity]="Active" **then**
278: **if** FindBufferUserCorrupt(user-id, Credential,roles) **then**
279: *creds* =Credential;
280: **else**
281: Credential= Generate credential for the user;
282: *creds* = Credential;
283: StoreBufferUserCorrupt(user-id,Credential,roles);
284: **end if**
285: Extract $creds_{service}$ from ServiceCorrupt(Service, Service-id, $creds_{service}$) where Service=$\mathcal{F}_{Image}$;
286: UpdateBuffer({Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid});
287: $C_3ID_5-$ Send-Env ($\mathcal{F}_{Image}$, Adversary, "Service validation, session-id, $creds_{service}$, *creds*, Request="Get Image image-id ");
288: **end if**

289: **else if** FindBuffer(Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid) **then**
290: **if** Corrupt[Identity]="Active" **then**
291: **if** Image-valid=0 **then**
292: UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid}, {Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, Fail});
293: $C_3ID_7-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$
294: **else**
295: UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid},{Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid})
296: $C_3ID_7-$ Send-IdealCloud("Get Image", session-id, image-id);%GOTO $C_3ID_8$
297: **end if**
298: **end if**
299: **else if** FindBuffer(Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid) **then**
300: **if** Corrupt[Identity]="Active" & valid=0 **then**
301: UpdateBuffer({Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid}, {Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, Fail});
302: $C_3ID_{10}-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$
303: **else if** Corrupt[Identity]="Active" & valid=1 **then**
304: UpdateBuffer({Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid}, {Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid});
305: $C_3ID_{10}-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);%GOTO $C_3ID_{11}$
306: **end if**
307: **else if** FindBuffer($\mathcal{F}_{Compute}$, destination, message) where the request-ID in the message is equal to session-id **then**
308: **if** destination=$\mathcal{F}_{Identity}$ **then**
309: **if** Corrupt[Identity] !="Active" **then**
310: valid=Fail;
311: **if** (user-id,*creds*,roles) $\in$ BufferUserCorrupt & request in the message $\in$ roles **then**
312: valid=Successful;
313: **end if**
314: UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message}, {Receiver=destination, Sender= $F_{Compute}$, "Service validated", session-id, Request, valid});
315: $C_3C_1ID_1-$ Send-Env(Receiver, Sender, "Service validated", Request, valid); % GOTO $C_3C_1ID_2$
316: **else**
317: Send-Env($\mathcal{F}_{Compute}$, Adversary, message);
318: **end if**

319:

320: **else if** destination=$\mathcal{F}_{Image}$ **then**

321: **if** Corrupt[Image] !="Active" **then**

322: **if** (user-id,*creds*) ∈ BufferUserCorrupt **then**

323: extract user-id from the message;

324: Message= message that the user credential is replaced by user-id;

325: UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message},{"Result", $\mathcal{F}_{Compute}$, destination, message});

326: $C_3C_1I_1-$ Send-IdealCloud(Message);%GOTO $C_3C_1I_2$

327: **else**

328: UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message}, {$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, NULL, Fail});

329: $C_3C_1I_1-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, FImage-id, user-id, "Get Image image-id ");%GOTO $C_3C_1I_4$

330: **end if**

331: **else**

332: Send-Env($\mathcal{F}_{Compute}$, Adversary, message);

333: **end if**

334: **end if**

335: **else if** FindBuffer(Receiver, Sender, "Service validated", session-id, Request, valid) **then**

336: **if** Sender=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" **then**

337: $C_3C_1ID_2-$ Send-Env(Receiver, Adversary, message);

338: **else if** Sender=$\mathcal{F}_{Image}$ & Corrupt[Image]="Active" **then**

339: $C_3I_7ID_2-$ Send-Env(Receiver, Adversary, message);

340: **end if**

341: **else if** FindBuffer($\mathcal{F}_{Image}$, destination, message) where the session-ID in the message is equal to session-id **then**

342: **if** destination= $\mathcal{F}_{Compute}$ & message=("Image", session-id, image-id, Corresponding-Image ) & FindBuffer(Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image") where the message request-ID=session-id **then**

343: UpdateBuffer({Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image});

344: RemoveBuffer($\mathcal{F}_{Image}$, destination, message);

345: $C_3I_7-$ Send-Env ($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);

346: **else if** destination=$\mathcal{F}_{Identity}$ **then**

347: **if** Corrupt[Identity] !="Active" **then**

348: **if** FindDBRequest(session-id, Credential, ServiceObject-List, Black-List) where request-ID=session-id & Credential=*creds* & ($\mathcal{F}_{Image}$,Object) ∈ ServiceObject-List & $\mathcal{F}_{Image} \notin$ Black-List **then**

349: AddBlack-List$_{\text{session-id}}$($\mathcal{F}_{Image}$);

350: valid=successful;

**else**
valid=fail;
**end if**
StoreBuffer(Receiver=destination, Sender= $\mathcal{F}_{Image}$, "Service validated", session-id, Request, valid);
$C_3I_7ID_1-$ Send-Env(Receiver, Sender, "Service validated", Request, valid);
**else**
Send-Env($\mathcal{F}_{Image}$, Adversary, message);
**end if**
**end if**
**else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid") **then**
UpdateBuffer({$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid"},{$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid"});
$C_3C_1I_4-$ Send-Env ($\mathcal{F}_{Identity}$, $\mathcal{F}_{Image}$, "Service Validated", session-id, user-id, service-id, Request="Get Image image-id ", valid);
**else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid") **then**
UpdateBuffer({$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid"}, {$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid"});
$C_3C_1I_5-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-idimage-id, Corresponding-Image);
**else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid") **then**
RemoveBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid");
**if** Corrupt[Compute]="Active" **then**
$C_3C_1I_6-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, message);
**end if**
**else if** FindBuffer("Output", Service, destination, message) **then**
RemoveBuffer("Output", Service, destination, message);
**if** Service=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" & message contains "Delete Node" **then**
$D_3O_1-$ Send-IdealCloud(Receiver, "Delete Node", session-id, node-id, valid);
**else if** Service=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" & message contains "Create Node" **then**
$C_3C_1O_1-$ Send-IdealCloud(Receiver, "Create Node result", session-id, node-id, valid);
**end if**
**else if** FindBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image") **then**
UpdateBuffer({Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image"},{Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image"})

$C_3I_3-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, Compute-id, "Create Node", Successful);
**else if** FindBuffer(Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image") **then**
UpdateBuffer({Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image"})
$C_3I_4-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id);
**else if** FindBuffer(Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image") **then**
UpdateBuffer({Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"});
**if** Corrupt[Image]="Active" **then**
**if** FindBufferUserCorrupt(user-id, Credential) **then**
*creds* =Credential;
**else**
Credential= Generate credential for the user;
*creds* =Credential;
StoreBufferUserCorrupt(user-id,Credential);
**end if**
$C_3I_5-$ Send-Env($\mathcal{F}_{Image}$, Adversary, message = ("Get Image", session-id, *creds*, image-id));% GOTO $C_3I_6$ FOR FORWARDING, $C_3I_6^*$ FOR MAKEING NEW REQUEST
**end if**
**else if** FindBuffer(Receiver, "Create Node Output", session-id) **then**
RemoveBuffer(Receiver, "Create Node Output", session-id);
$C_3I_{11}-$ Send-IdealCloud("Create Node Output", session-id);
**else if** FindBuffer(Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist) **then**
$D_3ID_6-$ Send-IdealCloud(Receiver, "Delete Node", session-id, node-id, valid); %GOT0 $D_3O_2$
**else if** FindBuffer(Receiver,"Delete Node", session-id, user-id, node-id) **then**
**if** Corrupt[Compute]="Active" **then**
**if** FindBufferUserCorrupt(user-id, Credential) **then**
*creds* =Credential;
**else**
Credential= Generate credential for the user;
*creds* =Credential;
StoreBufferUserCorrupt(user-id,Credential);
**end if**
$D_3C-$ Send-Env(Receiver, Adversary, "Delete Node", session-id, node-id, *creds*);
**else if** Corrupt[Identity]="Active" **then**
UpdateBuffer({Receiver,"Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id });
$D_3ID-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Delete Node");
**else**

416: $D_3-$ Send-IdealCloud("Confirm", session-id);
417: **end if**
418: **else if** FindBuffer(Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id) **then**
419: **if** Corrupt[Identity]="Active" **then**
420: **if** FindBufferUserCorrupt(user-id, Credential,roles) **then**
421: *creds* =Credential;
422: **else**
423: Credential= Generate credential for the user;
424: *creds* =Credential;
425: StoreBufferUserCorrupt(user-id,Credential,roles);
426: **end if**
427: **if** FindBuffServiceCorrupt(Service-id, Credential) **then**
428: $creds_{service}$ =Credential;
429: **else**
430: Credential= Generate credential for the service;
431: $creds_{service}$ =Credential;
432: StoreBufferServiceCorrupt(Service-id,Credential);
433: **end if**
434: UpdateBuffer({Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id });
435: $D_3ID_1-$ Send-Env(Receiver, Adversary, "Service Validation", session-id, $creds_{service}$, *creds*);
436: **end if**
437: **else if** FindBuffer(Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid) **then**
438: **if** Corrupt[Identity]="Active" **then**
439: **if** valid=0 **then**
440: UpdateBuffer({Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid}, {Receiver,"Corrupt Output Delete Node", session-id, user-id, node-id, valid});
441: $D_3ID_3-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, Fail);% GOTO $D_3ID_4^*$
442: **else**
443: RemoveBuffer(Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid);
444: $D_3ID_3-$Send-IdealCloud("Delete Node", session-id, node-id, valid); %GOTO $D_3ID_4$
445: **end if**
446: **end if**
447: **else if** FindBuffer(Receiver,"Corrupt Output Delete Node", session-id, user-id, node-id, valid) **then**
448: $D_3ID_4^*-$ Send-IdealCloud(Receiver,"Delete Node", session-id, node-id, valid);%GOTO $D_3O_2$
449: **else if** FindBuffer(Receiver, "Node Access", session-id, user-id, node-id, RequestActivity) **then**
450: Send-IdealCloud("Result Access Node", session-id)
451: **else if** FindBuffer(Receiver, "Node Access", session-id, user-id, node-id, result, valid) **then**
452: Send-IdealCloud("Output Access Node", session-id)
453: **else if** FindBuffer(Receiver, "Attach Volume", session-id, volume-id, node-id) **then**

454: **if** Identity is Corrupt **then**
455: valid = validity setting used by simulated adversary
456: **else**
457: valid=False
458: **end if**
459: Send-IdealCloud("Confirm Attach Volume", session-id, valid)
460: **else if** FindBuffer("Attach Volume", session-id, volume-id, node-id, valid) **then**
461: Send-IdealCloud("Result Attach Volume", session-id)
462: **else if** FindBuffer("Attach Volume", session-id, volume-id, node-id, success/fail) **then**
463: Send-IdealCloud("Output Attach Volume", session-id)
464: **else if** FindBuffer(Receiver, "Detach Volume", session-id, volume-id, node-id) **then**
465: **if** Identity is Corrupt **then**
466: valid = validity setting used by simulated adversary
467: **else**
468: valid=False
469: **end if**
470: Send-IdealCloud("Confirm Detach Volume", session-id, valid)
471: **else if** FindBuffer("Detach Volume", session-id, volume-id, node-id, valid) **then**
472: Send-IdealCloud("Result Detach Volume", session-id)
473: **else if** FindBuffer("Detach Volume", session-id, volume-id, node-id, success/fail) **then**
474: Send-IdealCloud("Output Detach Volume", session-id)
475: **else if** FindBuffer(Receiver, "Get Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image) **then**
476: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image},{Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image});
477: Send-Env ($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);
478: **else if** FindBuffer(Receiver, "Delete Service Validation", session-id, Service-id, user-id, node-id, valid) **then**
479: UpdateBuffer({Receiver, "Delete Service Validation", session-id Service-id, user-id, node-id, valid},{Receiver, "Delete Service Validated", session-id, Service-id, user-id, node-id, Valid})
480: $D_6-$ Send-Env ($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, service-id, Request, valid);
481: **else if** FindBuffer(Receiver, "Delete Service Validated", session-id, user-id, node-id, valid) **then**
482: UpdateBuffer({Receiver, "Delete Service Validated", session-id, user-id, node-id, valid},{Receiver, "Delete Node Continue", session-id, user-id, node-id, valid});
483: $D_7-$ Send-IdealCloud ("Delete Node Output", session-id, continue);
484: **else if** FindBuffer(Receiver, "Output Delete Node", session-id) **then**
485: $D_{10}-$ Send-IdealCloud("Output Delete Node", session-id);

**else if** FindBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image) **then**

UpdateBuffer({Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Service Validated", session-id, user-id, image-id, Service-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image})

$C_6-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, Compute-id, "Create Node", create-valid);

**else if** FindBuffer(Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image) **then**

**if** create-valid=successful **then**

UpdateBuffer({Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image})

$C_7I, C_7II-$ Send-Env ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id);

**else**

UpdateBuffer({Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Create Node Output", session-id });

$C_7III-$ Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, NULL, Fail);

**end if**

**else if** FindBuffer(Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image) **then**

UpdateBuffer({Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Service Validation", session-id, image-id, FImage-id, Image-valid, Corresponding-Image});

$C_8I, C_8II-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, FImage-id, user-id, "Get Image Image-id");

**else if** FindBuffer(Receiver, "Get Image Service Validation", session-id, image-id, FImage-id, Image-valid, Corresponding-Image) **then**

UpdateBuffer({Receiver, "Get Image Service Validation", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image})

$C_9I, C_9II-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Image}$, "Service Validated", session-id, user-id, Fimage-id, "Get Image image-id", Image-valid);

**else if** FindBuffer(Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image) **then**

504: **if** Image-valid =false **then**
505: Corresponding-Image=NULL
506: **end if**
507: UpdateBuffer({Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Image})
508: $C_{10}I, C_{10}II-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-id, image-id, Corresponding-Image);
509: **else if** FindBuffer(Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Image) **then**
510: **if** Corresponding-Image= NULL **then**
511: UpdateBuffer({Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Imaged},{Receiver, "Create Node Output", session-id })
512: $C_{11}II-$ Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, NULL, Fail);
513: **else**
514: UpdateBuffer({Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Imaged},{Receiver, "Request Node ID", session-id })
515:
516: $C_{11}I-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);
517: **end if**
518: **else if** FindBuffer(Receiver, "Create Node Output", session-id) **then**
519: RemoveBuffer (Receiver, "Create Node Output", session-id);
520: $C_8III, C_{12}II, C_{15}I-$ Send-IdealCloud ("Create Node Output", session-id);
521: **end if**
522:
523: $D_5-$ Upon receiving (Receiver, "Delete Node", session-id, user-id, node-id, valid) from Ideal Cloud:
524: Service-id=Service-id for $F_{Compute}$
525: StoreBuffer(Receiver, "Delete Service Validation", session-id, Service-id, user-id, node-id, valid);
526: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Delete node-id ");
527:
528: $D_9-$ Upon receiving ("Delete Node Completed", session-id, Valid, NodeExist) from Ideal-Cloud:
529: **if** FindBuffer(Receiver, "Delete Node Continue", session-id, user-id, node-id, valid) **then**

530: UpdateBuffer ({"Delete Node Continue", session-id, user-id, node-id, valid}, {"Output Delete Node", session-id })
531: **if** NodeExist=0 or Valid=0 **then**
532: Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-id, Fail);
533: **else**
534: Send-Env ($\mathcal{F}_{Compute}$, *Receiver,* "Delete Node", session-id, node-id, Successful);
535: **end if**
536: **end if**
537:
538: $C_5I-$ Upon receiving (Receiver, "Get Node ID", session-id, user-id, image-id, Corresponding-Image, Node-Structure) from Ideal Cloud:
539: StoreBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", Successful, "Get Image", Successful, Corresponding-Image);
540: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node Node-Structure");% GOTO $C_6$
541:
542: $C_5II-$ Upon receiving (Receiver, "Create Node", session-id, user-id, Fail, reason="Get Image", Node-Structure) from Ideal Cloud:
543: StoreBuffer(Receiver, "Service Validation", session-id, user-id, "Create Node", Successful, "Get Image", Fail, NULL);
544: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node Node-Structure");;% GOTO $C_6$
545:
546: $C_5III-$ Upon receiving (Receiver, "Create Node", session-id, user-id, Fail, reason="Create Node", Node-Structure) from Ideal Cloud:
547: StoreBuffer(Receiver, "Service Validation", session-id, user-id, "Create Node", Fail, "Get Image", Fail, NULL);
548: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node with Node-Structure");% GOTO $C_6$
549:
550: $C_{14}I-$ Upon receiving ("Create Node Continue", session-id, user-id, valid) from IdealCloud:
551: **if** FindBuffer(Receiver, "Node ID continue", session-id, node-id) **then**
552: UpdateBuffer({Receiver, "Node ID continue", session-id },{Receiver, "Create Node Output", session-id })
553: Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);
554: **end if**
555: **end function**

In more detail, we show the following three properties:

- **Property 1**: The simulator faithfully emulates the entire workflow required to process one user-provided command, when viewed standalone. We will demonstrate this property by analyzing the simulator's actions in response to a DeleteNode request; its response to other user actions are more cumbersome but similar in spirit.
- **Property 2**: The simulator faithfully emulates two interleaved user-provided commands such that the resulting state is the correct outcome after executing the two commands in an adversarially-controlled order. Put differently, there are no race conditions that cause the services to reach any kind of "weird state". We will explain this property by analyzing the simulator's actions when CreateNode and DeleteNode requests are interleaved.
- **Property 3**: An adversary who compromises services can only leverage them to damage other (uncompromised) services by leveraging OpenStack's token mechanism for authentication and authorization. Our ideal worlds accurately reflect the extent of the damage that the environment can cause by misusing acquired bearer tokens.

  We will showcase this property by considering an adversary who has compromised only Nova (and not any other services) and who is trying to glean as much information as possible from Glance. In section 5 and 6, we will show the analogous property using a stronger 'one-time' token mechanism, in which the simulator responds slightly differently when dealing with service-to-service interactions.

#### 4.4.1 Property 1: single command

By inspecting a sample command DeleteNode, we show that the view of the Environment $E$ is the same for the ideal and OpenStack Services world. For the ideal world, the simulator $S$ emulates the entire work flow that is required to process a single user-provided command, when viewed standalone.

In the OpenStack Services world, $E$ interacts with the cloud services through $\mathcal{F}_{ExtNet}$ and $\mathcal{F}_{SMT}$. To validate that the OpenStack Services doesn't leak more information than the ideal cloud, $S$ emulates the leakage of $\mathcal{F}_{ExtNet}$ and $\mathcal{F}_{SMT}$ in the OpenStack Services.

For the DeleteNode command, as illustrated in Figure 3, the following four type of messages are leaked through $\mathcal{F}_{ExtNet}$ and $\mathcal{F}_{SMT}$ (in the OpenStack Services) which the simulator $S$ is in charge to emulate them:

**Type-1:** A "Delete Node" message from $\mathcal{F}_{ExtNet}$ of the form ($U_{Dashboard}$, $\mathcal{F}_{Compute}$, "Delete Node", session-id, node-id, user-id) sent in Step 3 in Figure 3.

**Type-2:** A "Delete Node Validation" message from $\mathcal{F}_{SMT}$ of the form ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Delete node-id ") sent in Step 7 in Figure 3.

**Type-3:** A "Delete Node Validated" message from $\mathcal{F}_{SMT}$ of the form ($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, Compute-id, "Delete node-id ", valid) sent in Step 11 in Figure 3.

**Type-4:** A "Notification" message from $\mathcal{F}_{ExtNet}$ of the form ($\mathcal{F}_{Compute}$, $U_{Dashboard}$, "Delete Node", session-id, node-id, result) sent in Step 15 in Figure 3.

In OpenStack Services, when the deletenode command is executed, the first message that $E$ gets is the Type-1 message, which indicates the command that is requested by the user.

In response, the simulator $S$ acts as follows. Upon receiving a delete node "Request-message" from $\mathcal{F}_{Cloud}$, $S$ runs the Dashboard protocol of the user with user-id with the input-message "Request-message." This command activates $\mathcal{F}_{ExtNet}$ with ($U_{Dashboard}$, $\mathcal{F}_{Compute}$, "Delete Node", session-id, node-id, *creds*) and returns a Type-1 message to $S$. Then, $S$ forwards this message to $E$.

When $E$ hands back the control to the OpenStack Services by sending the message ("confirmation", session-id), the Compute service then sends a validation request to Identity service which is leaked to $E$ through $\mathcal{F}_{SMT}$ (Type-2 message).

In order to emulate this leakage so that $E$'s view is the same in both worlds, the simulator $S$ sends the environment's message back to the ideal cloud because it needs to know the validation result of the "Delete Node" request.

Upon receiving the reply from $\mathcal{F}_{Cloud}$ of the form ($U_{Dashboard}$, "Delete Node", session-id, user-id, node-id, valid), $S$ stores the message in its buffer with StoreBuffer(Receiver, "Delete Service Validation", session-id, Compute-id, user-id, node-id, valid). Using the message given by $\mathcal{F}_{Cloud}$, $S$ creates a Type-2 message and sends it to $E$. Note that the corresponding "Delete Node" message has already been buffered in $\mathcal{F}_{Cloud}$, which means that $\mathcal{F}_{Cloud}$ and $\mathcal{F}_{Compute}$ are in the same consistent state.

When $E$ hands back control to the OpenStack Services, the Identity service provides the validation result in the Type-3 format. This message is leaked through $\mathcal{F}_{SMT}$ to $E$.

For $S$ to create Type-3 message, it needs the validation result, which already exists in its buffer. Therefore, the simulator responds to $E$ with a Type-3 message based on the validation result that has been stored in its buffer.

Upon getting the control back from $E$, the node is deleted (if it exists) by Compute service and the result message (Type-4) is leaked to $E$ through $\mathcal{F}_{ExtNet}$.

However, in the ideal cloud, the node has not been deleted yet, this is why the simulator sends $\mathcal{F}_{Cloud}$ a message of the form ("Delete Node", session-id, Continue).

Upon receiving the reply from $\mathcal{F}_{Cloud}$ of the form ("Delete Node", session-id, NodeExist), $S$ updates the buffer with the node existence information. Note that $S$ does not send the node existence information, but uses it together with valid value to create Type-4 message. $S$ sets "result" as "Success" if the buffered variables Valid = 1 *and* NodeExist = 1, and sets as "Fail" otherwise. Then $S$ creates Type-4 message and sends it to $E$.

When $E$ hands back the control to the OpenStack Services, the user then receives the result message, thus $S$ sends a message to $\mathcal{F}_{Cloud}$ which enables $\mathcal{F}_{Cloud}$ to inform the user with the operation result.

For DeleteNode command, since the messages that leak from the ideal world/$S$ are the same as the messages that leak from the OpenStack Services, the environment cannot distinguish between these two worlds.

### 4.4.2 Property 2: multi command

In this section we explain how interleaving of two different user-provided commands maintains our invariant even when the adversary controls the order on intermediate message passing. More precisely, we analyze the simulator's actions precisely to verify that $E$ sees the same view in both worlds even when CreateNode and DeleteNode requests are interleaved.

In the OpenStack Services wold, the DeleteNode command consists of the following steps:

1. User sends a "Delete Node" request with a unique session-id and a node-id to $\mathcal{F}_{Compute}$.
2. $\mathcal{F}_{Compute}$ validates that the user has permissions to delete a node by asking $\mathcal{F}_{Identity}$.
3. If the user is able to delete the node, $\mathcal{F}_{Compute}$ makes sure that the node with node ID=node-id exists and if so, deletes the node.
4. $\mathcal{F}_{Compute}$ notifies the user about the result.

The CreateNode command consists of several steps as follows:

1. User sends a "Create Node" request with a unique session-id and required specification, such as image-id to $\mathcal{F}_{Compute}$.
2. $\mathcal{F}_{Compute}$ validates the user to create a node with the specified *specification* by asking $\mathcal{F}_{Identity}$.
3. If the user is allowed to create the node –the validation result is true, then $\mathcal{F}_{Compute}$ requests the corresponding image of image-id from $\mathcal{F}_{Image}$.
4. $\mathcal{F}_{Image}$ validates the user's permissions to extract the image corresponding to image-id by asking $\mathcal{F}_{Identity}$. It sends the result to $\mathcal{F}_{Compute}$.
5. If the user is allowed to get the corresponding image of image-id, $\mathcal{F}_{Compute}$ requests a node-id from adversary.
6. Adversary provides a node-id.
7. $\mathcal{F}_{Compute}$ makes sure that the node-id is not used, if so, creates a node using the corresponding image and the specification and assigns the node-id to it.
8. $\mathcal{F}_{Compute}$ notifies the user about the result.

We emphasize the following properties of the two processes. First, the DeleteNode process shows that the node-id is evaluated only after $\mathcal{F}_{Compute}$ is sure that the user is allowed to delete the node and that the node exists. Second, the CreateNode process permits the node to be created only after step 6 in which the adversary provides the node-id.

Let's consider the situation in which the user first runs the DeleteNode command for node-id =2 (which exists) then requests to "create a node" in which the adversary provides node-id =2 (in step 6 of the CreateNode command). We argue that, based on how the adversary may control the order of the intermediate messages, the result and $E$'s view of these two commands in both worlds (OpenStack Services and ideal world) are the same and the simulator may never get to an undefined state. Note that the result may be different based on the adversarially-chosen order of intermediate messages, but the point here is that the two worlds provide the same result for each possible message ordering.

In the OpenStack Services, if the requests are done in the given order, the node with node-id =2 should be deleted first and a new node with node-id =2 be created next. However, if the adversary prevents the delete node to be deleted and lets the create node complete first, then the node wouldn't be created because the node-id =2 is being used.

The simulator should also be able to act in the same way while not getting into an unexpected state. Note that validation of the user credentials and access list of one command doesn't affect the result of any other command. Therefore, we divide the CreateNode process into two phases.

**Phase 1:** Before getting node-id from the adversary (steps 1-5).

**Phase 2:** Getting node-id from the adversary and after that (steps 6-8). Note that in these steps, $\mathcal{F}_{Compute}$ executes without interruption from the adversary (or anyone else).

Phase 1 only deals with validation and getting corresponding image of image-id. During this phase, the ideal cloud provides the corresponding information by sending one of these two messages:

**Type-A:** (Receiver, "Get Node ID", session-id, user-id, image-id, Corresponding-Image) for the case where the user is able to create a node and get the corresponding image related to image-id or

**Type-B:** (Receiver, "Create Node", session-id, user-id, Fail, reason) for the case where the act of creating a node fails for reason = "user is not valid to create a node" or "user is not valid to get image related to image-id ".

If $S$ has received a Type-B message from the cloud, the CreateNode command has failed due to invalid user permissions. In this case, any ordering of CreateNode and DeleteNode may not result into unexpected state for $S$ because the validation messages have no effect on the result of any other message.

When the simulator receives a Type-A message, then the user is authorized to create a node and get the image related to image-id. Therefore, if all intermediate messages of *Phase 1* are sent, then the simulator can ask $E$ for a node-id (which is the beginning of *Phase 2*).

If $S$ receives a Type-A message, then the adversary can interleave messages in one of two ways:

1. The adversary might allow the DeleteNode command be completed before *Phase 2* of CreateNode command. In this case, the adversary has not given a node-id and thus the ideal cloud has not examined the usage of that node-id. Therefore, if the given node-id exists and the user is allowed to delete a node, the DeleteNode command will be done successfully and the create node can be completed with the same node-id.

2. The adversary might allow *Phase 2* of the CreateNode command to be completed before the DeleteNode command. In this case, the simulator has received the node-id for creating a node from the adversary. $S$ then passes this message immediately to the cloud. At this state, the ideal cloud examines the node usage and if it is used (which in our case its true) then the result of create node will be "Fail" but the node is able to be deleted next.

In summary: while the adversary's choice of ordering messages can influence the final state, in both circumstances the outcome is identical in the OpenStack Services and ideal cloud, which means that $E$'s view must also be identical in both worlds. In particular, the simulator never reaches a state that is not defined by the services, and the execution of its commands yields the same result as the OpenStack Services when the adversary controls the message ordering.

### 4.4.3 Property 3: service compromise

In this section, our aim is to showcase how the simulator acts when having a compromised service such that the environment $E$ sees the same view in both worlds.

Note that when a service is compromised, no matter which token mechanism is used, the adversary can do anything within the service's authority. Meanwhile, dealing with another service (when one of them is compromised) is not possible without using a token mechanism. Using the compromised service's capabilities, the adversary may try to deceive an uncompromised-service to get more information. In order to make this happen the adversary needs to send legitimate token with its request while dealing with *service-to-service* interaction. This is where the difference in security provided between the bearer token and the forthcoming, stronger one-time token model in Sections 5 and 6 can be demonstrated. For this reason, we will focus more on the service-to-service interactions and show how the simulator acts per each token models such that the environment $E$ gets the same view in both worlds. We exemplify this scenario by discussing the case in which the adversary has compromised only Nova and is trying to get as much information as possible from Glance.

We illustrate Property 3 with Nova as the compromised service because several bearer tokens pass through Nova that are intended for different services. Ergo, a compromised Nova offers a representative case study for capturing the amount of damage that an active adversary can do; any other service-to-service interaction involves comparable or simpler reasoning.

In general, when Nova is compromised, the adversary gets access to credentials corresponding to the following:

- Any actions that users make of $\mathcal{F}_{Compute}$ while it is compromised.
- Any actions that users requested of $\mathcal{F}_{Compute}$ before it was compromised but that remained unfulfilled before the compromise.

Recall that in the ideal world there are no credentials; ergo, it is the simulator's responsibility to create, store, and track any compromised credentials as needed so that $E$ sees the same result from both worlds for both models. Below, we consider the simulator's responsibility in each of the two token models.

**Property 3 in the bearer token model** The bearer token has two properties: 1- Unscoped, 2- No limitation on the number of usage. In the OpenStack Services, because of the first property, the adversary may make different requests to different services using the compromised service and user-id. And because of the second property, the adversary may use the compromised user-id and send a request to a service more than once.

The simulator should represent these properties in any condition, specially when a service is compromised, such that the environment $E$ gets the same view from both worlds. To realize these properties, the simulator creates a credential per each compromised user (user-id). That is, when Nova is compromised, the simulator notifies the ideal cloud and the ideal cloud sends all user-id (together with their roles) that have made either a new request or have an unfinalized request while Nova was corrupted. Then, the simulator creates a unique credential for each user-id, stores it in its database for future use, and adds these credentials to the corresponding messages as needed. Having these credentials however, the simulator is not able to respond to the adversarial validation

request for a compromised user-id because in this case the simulator does not have the list of roles assigned to the manipulated user.

As an example, let's consider that Nova is compromised while user with user-id =5 has uncompleted request and this user is able to have access to images related to image-id =2,10. In this situation, if the adversary/$E$ makes the following request to Glance (in the openstack world): "GetImage with image-id =2 for user-id =5", the Glance sends a request to Identity to validate user access and receives back the response. Based on the validation result, Glance provides the corresponding image to Compute. In the ideal world, the simulator is responsible to create these intermediate messages. Simulator has answer to neither Glance's validation request nor image, therefore it should ask the ideal cloud.

The simulator first makes sure that the user credential related to user-id =5 is correct by comparing it with the information in its database, then sends a request to the ideal cloud to get the validation result and the image related to the image-id =2. Based on the ideal cloud's response, the simulator creates the corresponding messages and sends it to $E$.

Because of "unlimited time of use" property that bearer token has, the adversary is able to repeat this request at any time. Which means all image information associated with image-id =2 for the rest of time should be considered compromised. In addition, due to the unscoped property, the simulator only stores a credential per user-id and therefore, the adversary is able to make another request to Glance to extract the image related to image-id =10 successfully (if s/he uses the same credential). Hence, $E$'s view will be identical in both worlds while using bearer token mechanism.

# 5 Improved Tokening Mechanism

Our analysis has shown that the simulator of the ideal cloud cannot emulate the corrupted services unless we relax the security promises of the ideal cloud. These concerns arise due to the use of bearer tokens, which enable anyone who can see a token to masquerade as corresponding user and perform any action allowed to the owner of the token on behalf of him. Bearer-tokens are leaked to the adversary in both partially and fully corrupted services which could happen through a vulnerability. Once a service corrupted, it could be malicious in at least two important ways:

- A service can use a token to obtain unrequested services charged to the user.
- A service can use a token to access other resources at another service (like exporting a disk with sensitive information).

The above problems are not tolerated by the cloud users. Having a bug free cloud is not realistic, but reducing effects of buggy code is possible. Therefore, it is desired to have OpenStack guarantee the modularity security requirement defined as follows.

**Modularity Security Requirement** If an adversary corrupts a service, other services just do whatever requested to do by the users, nothing more. For example, if an adversary corrupts the Image service, he cannot create or delete a node, but if a user issues a "create node" command, it is possible to create a node with a corrupted image because of the image service corruption.

We propose to limit the impact of service corruption by preventing replay attacks and scoping tokens only to handle the request desired by the user (following an idea laid out in [24, 25]). As a result, even if an attacker captures a credential submitted to a non-Keystone service by an honest user, he cannot do anything more on behalf of the user. In this section, we explain our two proposals for the OpenStack token issue which improve current security guarantees: 1) Ticketing System, and 2) Recursive Augmented Fernet Tokens. Then, in Section 6 we analyze OpenStack with our proposed scheme in order to demonstrate rigorously that our token mechanism realizes our goal of providing security in the face of service corruptions.

### 5.1 Ticketing System

A one-time use token scoped to a specific job resembles a "ticket" as used in Kerberos [42] or in TLS 1.3's zero round trip time mechanism [43]. A ticket cannot be used more than one time and it is not possible to use it for any arbitrary action. Therefore, one option can be to replace the current bearer-token mechanism with a ticketing system as follows:

1. User asks Keystone for a ticket scoped to a specific job (e.g. delete a node, attach a volume, upload an image). Additionally, the user with the help of Dashboard protocol provides the restriction list required for the unique job such as node Id, volume Id, hash of the image, and the list of services that will be involved in the job processing steps.

2. Keystone creates a ticket for the request, stores the ticket and the list of the services required for the request in its DB, and sends the ticket to the user. Tickets are supposed to be used immediately by users. Therefore, they have a very short expiration time and the DB can be implemented in ephemeral memcached memory.

3. The User sends the ticket to the first service.

4. The first service asks Keystone to validate the ticket.

5. If the ticket and the name of the service exist in the DB, Keystone removes the name of service from the corresponding service list and returns True. Otherwise, it returns False. If the list of services is empty, the ticket is removed from the keystone's memory.

The ticketing system is specified fully in Algorithm 27.

**Pros and cons.** The proposed ticketing system is a simple and straightforward mechanism for providing the modularity security requirement. It keeps the token (ticket) size small. However, the user needs to have an extra communication with Keystone to get a ticket for each request. Hence, a major change is needed for all the user interfaces.

### 5.2 RAFTs

The main drawback of the ticketing system is that a major change must be made to OpenStack to implement it. Here, we propose a new, backwards-compatible Fernet extension that we call Recursive Augmented Fernet Tokens (RAFTs), on top of which user-specified restrictions can float independently without requiring additional Keystone interaction. The RAFT token mechanism provides the modularity security requirement in the presence of one or more corrupted services just as ticketing accomplished. Fundamentally, RAFTs use the Fernet HMAC as a shared secret between the user and Keystone that can be used to authenticate arbitrary user-supplied restrictions.

**Algorithm 27** Ticketing System

```
1: function IDEAL IDENTITY
2:     Upon receiving (creds, Job-Type, Restriction-List) from User U:
3:     if creds is valid then
4:         Create ticket for the Job using {Job-Type, Restriction-List, U};
5:         StoreDB(ticket, Service-List);
6:         return ticket;
7:     else
8:         return "fail";
9:     end if
10:
11:    Upon receiving (F_Service, F_Identity, "Validate", ticket) from F_Service:
12:    if FindBD(ticket, Service-List) where the ticket is valid and F_Service is in the Service-List
   then
13:        New-Service-List=Remove F_Service from the Service-List;
14:        if New-Service-List is Empty then
15:            RemoveDB(ticket, Service-List);
16:        else
17:            UpdateDB({ticket, Service-List }, {New-Service-List, ticket });
18:        end if
19:        Send-F_Service (F_Identity, session-id, Successful);
20:    else
21:        Send-F_Service (F_Identity, session-id, Fail);
22:    end if
23: end function
24: function SERVICE
25:    Upon receiving (Sender, ticket) from user U:
26:    StoreBuffer (Sender, ticket);
27:    Send-F_Identity (F_Service, "Validate", ticket);
28:
29:    Upon receiving (F_Identity, session-id, valid) from F_Identity:
30:    if valid=1 & FindBuffer(Sender, ticket) where session-ID in the Job-Request=session-id
   then
31:        Apply the Request;
32:    else
33:        Return Fail;
34:    end if
35: end function
```

The Fernet token is a recent innovation that uses cryptography to provide authenticity without accessing a central DB. It is a mechanism by which Keystone creates a private, authenticated channel to itself. It has quickly become the preferred token format for OpenStack as it does not require maintaining a central database of valid tokens, which would add network load and complexity when running a distributed Keystone service. The current Fernet token mechanism is [44]. The Fernet token is represented as follows:

$$\begin{aligned} Sk &= \text{Keystone signing key} \\ Token_{Fernet} &= Specification \parallel HMAC_{Sk}(Specification) \end{aligned}$$

Because the original Fernet token is a bearer one, the link between the user and Keystone must be authenticated and encrypted to ensure that only the user receives $Token_{Fernet}$ from Keystone.

In RAFT, a user first obtains a Fernet token from Keystone. The user takes the $HMAC_{Sk}(Specification)$ as a shared secret key $Uk$ between Keystone and himself. Then, for each of his subsequent requests, the user calculates:

$$\begin{aligned} Token_{RAFT} = & Request \parallel \\ & Specification \parallel \\ & Restrictions \parallel \\ & HMAC_{Uk}(Request \parallel Specification \parallel Restrictions) \end{aligned}$$

The *Restrictions* could be a composition of anything like a shortened expiration time ("expiration-time = 5 secs") or endpoint restrictions ("service=cinder"). The RAFT token is a self-explained token and the user request is part of the token and there is no need to accompany the request with the token. Hence, the user just sends the RAFT token to the desired service. For the sake of backward compatibility, which we will talk about later, all the services continue to accept job requests in the format of $(Request, Token)$ where either $Token = Token_{RAFT} - Request$ or $Token = Token_{Fernet}$.

When a service receives $Token_{RAFT}$, the service forwards it to Keystone asking for validation. If the answer of Keystone was "valid", the service extracts the *Request* and *Restrictions* from the token and gets the work done accordingly.

In the current OpenStack architecture, each service gets a service token from Keystone. The service token is used to authenticate the service to other services [45]. RAFT replaces this bearer token as well: the token of a service is never sent to other services, but is instead used as a signing key $Svk$ in the RAFT framework. Concretely, if a service needs to involve other services, then it calculates:

$$\begin{aligned} Svk = & \text{Service signing key, from Keystone} \\ New\_Token_{RAFT} = & Token_{RAFT} \parallel \\ & Restrictions \parallel \\ & ServiceId \parallel \\ & HMAC_{Svk}(Token_{RAFT} \parallel Restrictions), \end{aligned}$$

and sends the result to the required services. We highlight three tangible benefits of this method: each service is able to add (but not remove!) *Restrictions* to the user's token, the MAC authenticates

the channel used to make a service-to-service request, and the token provides the recipient service with the provenance of the request [45].

When Keystone receives a token validation request from a service $S$ for token $T$, it follows the Fernet protocol as if $T$ is a Fernet token. Keystone looks into its blacklist to see whether $T$ was used by $S$ or not. If yes, Keystone returns fail. Otherwise, Keystone validates all the service signatures. Then, it takes out *Specification* from $T$ and calculates $HMAC_{Sk}(\textit{Specification})$ as its signing key. Using the result, Keystone is able to verify the user's signature embedded into $T$, too. If the token is valid, Keystone adds $(T, S)$ to its blacklist and returns "valid". Just as in the ticketing system, the expiration time of RAFT tokens are very short. Therefore, the blacklist can be implemented in a lightweight memcached structure.

The RAFT is specified fully in Algorithm 28.

**Pros and cons.** RAFT uses the Fernet token to create a user to keystone private/authenticated channel, while not jeopardizing keystone's existing channel to itself. It can be implemented in a backward compatible way. RAFT tokens are larger than Fernet tokens and the size of buffers used for storing tokens in different OpenStack modules must be carefully inspected.

### 5.3 Effect of One-Time Tokens on Corruption

Restricting tokens to be single use with specific parameters limits the scope of adversarial corruptions. Previously a corrupted service could store all tokens it sees and use them at will. This modification ensures that once a token has been used it will not be accepted again in the future. Additionally, since tokens are scoped, it is not possible for a corrupted service to lie to uncorrupted services about the content of a user's request. Specifically, users will indicate when requesting a token exactly which of their resources it grants access to. This would not prevent a corrupted Compute from deleting a node when a user had asked to create one or creating a node with a completely different specification since Compute could simply do these things without bothering to verify the token, but it *would* prevent Compute from requesting a different image from the Image service than the one the user had intended. This helps in the case where the adversary has not already seen, via a request by a different user, the image it would like to have Compute use. Scoped, single use tokens even benefit functionalities that go beyond those captured in our model; for example, an adversary who corrupted the Compute service would be unable to upload arbitrary images.

We also remark that there is another notion of limited-use tokens with a security-performance tradeoff. Many OpenStack engineers desire a stateless Keystone architecture in order to distribute the authentication process over several servers and improve scalability and performance. If it is desirable to keep Keystone stateless, then one could instead design "limited-reuse" tokens with a short lifetime. While one could model the impacts of limited reuse by combining rate limiting with a UC model of network time [26], in this paper we focus on the One-Time token presented above with an explicit blacklist to prevent reuse.

Juxtaposing the ideal clouds in the bearer token (Algorithm 8) and One-Time token (Algorithm 29) scenarios is quite illustrative. An OpenStack developer who might not understand the full complexity of UC (and thus who might ignore our simulators and proofs) can still compare those two algorithms to understand the harms of partial compromise in the two settings. For example, Algorithm 8 shows that a corrupted $\mathcal{F}_{BlockStorage}$ can tamper with the delivery of an image from $\mathcal{F}_{Image}$ to $\mathcal{F}_{Compute}$ during a Create Node operation in the bearer token model, whereas Algorithm 29 shows that the same attack is not possible in the ideal cloud for the One-Time token model.

**Algorithm 28** Recursive Augmented Fernet Token

**function** IDEAL IDENTITY
    Upon receiving (*creds*, TokenRequest) from User U:
    **if** *creds* is valid **then**
        Create *Token* using the specification of U stored in the Keystone;
        return *Token*;
    **else**
        return "fail";
    **end if**

    Upon receiving ($\mathcal{F}_{Service}$, $\mathcal{F}_{Identity}$, "Validate", *Token*) from $\mathcal{F}_{Service}$:
    **if** *Token* is in Fernet format **then**
        Invoke Fernet-Validation-Protocol;
    **else**
        **if** *Token* is in RAFT format **then**
            try{
            assert(*Token* ExpirationTime− $\mathcal{F}_{Service}$ Allowance < current-time);
            assert(All service signatures inside *Token* are valid);
            *Specification*= Extract-*Specification*(*Token*);
            *Request*= Extract-*Request*(*Token*);
            $Uk = HMAC_{SK}(Specification)$ ;
            $UserSign = HMAC_{UK}(Request \parallel Specification)$;
            $UserSign'$ = Extract-*UserSign*(*Token*);
            assert($UserSign = UserSign'$);
            assert(FindDB(*Token*,$\mathcal{F}_{Service}$)=NULL);
            Send-$\mathcal{F}_{Service}$ ($\mathcal{F}_{Identity}$, session-id, Successful);
            }catch{
            Send-$\mathcal{F}_{Service}$ ($\mathcal{F}_{Identity}$, session-id, Fail);
            }
        **end if**
    **end if**
**end function**
**function** SERVICE
    Upon receiving (Sender, *Token*) from user U:
    *Request*= Extract-*Request*(*Token*);
    **if** Sender is allowed to do *Request* **then**
        StoreBuffer (Sender, *Token*, session-id);
        Send-$\mathcal{F}_{Identity}$ ($\mathcal{F}_{Service}$, "Validate", *Token*);
    **end if**

    Upon receiving ($\mathcal{F}_{Identity}$, session-id, valid) from $\mathcal{F}_{Identity}$:
    **if** valid="Successful" & FindBuffer(Sender, *Token*,session-id) **then**
        Apply the Request;
    **else**
        Return Fail;
    **end if**
**end function**

47: **function** DASHBOARD
48: Upon receiving (Sender, *Token*) from Keystone:
49: $Uk$ = Extract-Token-Sign(*Token*) ;
50: *Specification*= Extract-*Specification*(*Token*);
51:
52: Upon receiving ($\mathcal{F}_{Service}$, *Request*, *Restrictions*) from user U:
53: $Specification' = Request \| Specification \| Restrictions$;
54: $Token = Specification' \parallel HMAC_{UK}(Specification')$;
55: Send-$\mathcal{F}_{Service}$ (U, *Token*);
56: **end function**

In the next section, we formalize this intuition and demonstrate that OpenStack with a one-time token model UC-realizes a stronger ideal cloud.

More generally, the UC ideal cloud that models any suggested security improvement permits OpenStack developers to understand concretely the value added by this improvement. This is an example for how UC modeling can provide developers with a valuable new tool that they can use to balance the relative importance of addressing any of the numerous bug reports on their plate at any given time.

# 6 UC Analysis of OpenStack with Improved Tokening Mechanism

In this section we aim to analyze the security of the strengthened version of OpenStack in the universal composability setting. Particularly, we want to prove the following theorem:

**Theorem 3.** *The strengthened version of OpenStack Services using RAFT securely realizes the less-leaky version of $\mathcal{F}_{Cloud}$ in Algorithm 29 (in section 6.1) in the ($\mathcal{F}_{ExtNet}$, $\mathcal{F}_{SMT}$)-hybrid model.*

For this reason, we need to formaly specify the ideal cloud and strengthened version of OpenStack Service in UC syntax, provide a simulator, and alalyze that the simulator works to insure that the views of the environment is the same in both worlds. The rest of this section covers these purposes.

Note that all description and assumption made in section 3 holds for these algorithms.

## 6.1 Ideal Cloud Functionality Improved Tokens

We provide the ideal cloud functionality for the scenario with improved tokens in Algorithm 29 [5].

[5]The description of notation used in the algorithm is provided in appendix A

**Algorithm 29** Ideal Cloud with corrupted services for Improved Token

1: **function** CLOUD
2:
3: Upon receiving ("Corrupt", Service, value) from Sim:
4: Corrupt[Service]=value; % the default value=None. None indicates no corruption; Passive is the partial corruption, and Active is the full corruption
5: **if** Service=$\mathcal{F}_{Image}$ & value!=None **then**
6: **if** there are user requests that have not finalized **then**
7: Add (user-id) to the List;
8: **end if**
9: send-Sim(DB of (image-id, corresponding-Image),List of (user-id));
10: **else if** Service=$\mathcal{F}_{Compute}$ & value!=None **then**
11: **if** there are user requests that have not finalized **then**
12: Add (user-id) to the List;
13: Send-Sim(List of (user-id));
14: **end if**
15: **else if** Service=$\mathcal{F}_{Identity}$ & value!=None **then**
16: Send-Sim(DB of (user-id, roles));
17: **else if** Service=$\mathcal{F}_{BlockStorage}$ & value!=None **then**
18: **if** there are user requests that have not finalized **then**
19: Add (user-id) to the List;
20: **end if**
21: Send-Sim(List of (user-id));
22: **else if** Service=$\mathcal{F}_{Node}$node-id & value!=None **then**
23: **if** there are user requests that have not finalized **then**
24: Add (user-id) to the List;
25: **end if**
26: Send-Sim(List of (user-id));
27: **else if** Service=$\mathcal{F}_{Volume}$volume-id & value!=None **then**
28: **if** there are user requests that have not finalized **then**
29: Add (user-id) to the List;
30: **end if**
31: Send-Sim(List of (user-id));
32: **end if**
33: Upon receiving (Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity) from user U:
34: **if** session-id =(U,N) where N non-repetitive for user U **then**
35: StoreBuffer (Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity);
36: **end if**
37: Send-Sim (Receiver, "Node Access", session-id, user-id, node-id, RequestActivity);
38:

39: Upon receiving("Result Access Node", session-id) from Sim:
40: **if** FindBuffer(Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity, valid) **then**
41: **if** valid=True **then**
42: results ← EXECUTE(AccessRequest);
43: UpdateBuffer({Receiver, "Access Node", session-id, *creds*, node-id, RequestActivity, valid}, {Receiver, "Access Node", session-id, *creds*, node-id, result, valid});
44: Send-Sim ("Access Node", session-id, user-id, node-id, result, valid);
45: **end if**
46: **end if**
47:
48: Upon receiving("Output Access Node", session-id) from Sim:
49: **if** FindBuffer(Receiver, "Access Node", session-id, *creds*, node-id, results, valid) **then**
50: Valid=valid;
51: RemoveBuffer("Access Node", session-id, *creds*, node-id, results, valid);
52: Output("Access Node", session-id, results, Valid) to *Receiver*;
53: **end if**
54:
55: Upon receiving ("Access Node", session-id, node-id, RequestActivity) from Sim:
56: **if** Corrupt[Node node-id ]=Active **then**
57: results ← EXECUTE(AccessRequest);
58: **else**
59: results = None
60: **end if**
61: Send-Sim("Access Node", session-id, node-id, RequestActivity, results)
62:
63: Upon receiving (Receiver, "Attach Volume", session-id, *creds*, volume-id, node-id) from user U;
64: **if** session-id =(U,N) where N non-repetitive for user U **then**
65: StoreBuffer(Receiver, "Attach Volume", session-id, *creds*, volume-id, node-id);
66: **end if**
67: Send-Sim(Receiver, "Attach Volume", session-id, volume-id, node-id);
68:
69: Upon receiving ("Result Attach Volume", session-id) from Sim:
70: **if** FindBuffer(Receiver, "Attach Volume", session-id, *creds*, volume-id, node-id, valid) **then**
71: **if** valid=True AND attached-volume for node node-id is None AND attached-node for volume volume-id is None **then**
72: Set attached-volume for node node-id to volume-id

```
73:             Set attached-node for volume volume-id to node-id
74:             UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id,
    valid) }, {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success })
75:             Send-Sim("Attach Volume", session-id, volume-id, node-id, success)
76:         else
77:             UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id,
    valid) }, {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure) })
78:             Send-Sim("Attach Volume", session-id, volume-id, node-id, failure)
79:         end if
80:     end if
81:
82:     Upon receiving ("Output Attach Volume", session-id) from Sim:
83:     if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success)
    then
84:         RemoveBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, success)
85:         Output("Attach Volume", session-id, success) to Receiver
86:     else if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure)
    then
87:         RemoveBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id, failure)
88:         Output("Attach Volume", session-id, failure) to Receiver
89:     end if
90:
91:     Upon Receiving (Confirm, "Attach Volume" session-id, valid)
92:     if FindBuffer(Receiver, "Attach Volume", session-id, creds, volume-id, node-id) then
93:         if user user-id is allowed to attach volume-id to node-id for session session-id then
94:             valid=True
95:         else
96:             valid=False
97:         end if
98:         if Corrupt[Identity]=Active then
99:             valid = value sent by Sim
100:        end if
101:        UpdateBuffer({Receiver, "Attach Volume", session-id, creds, volume-id, node-id },
    {Receiver, "Attach Volume", session-id, creds, volume-id, node-id, valid })
102:        Send-Sim(Receiver, "Attach Volume", session-id, volume-id, node-id, valid)
103:    end if
104:
105:    Upon receiving ("Attach Volume", session-id, volume-id, node-id) from Sim;
106:    if Corrupt[Compute]=Active OR Corrupt[Node node-id ]=Active then
107:        Set attached-volume for node node-id to volume-id
108:    end if
```

109: **if** Corrupt[Storage]=Active OR Corrupt[Volume volume-id ]=Active **then**
110: Set attached-node for volume volume-id to node-id
111: **end if**
112: Send-Sim ("Detach Volume", session-id, volume-id, node-id);
113: Upon receiving (Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id) from user U;
114: **if** session-id =(U,N) where N non-repetitive for user U **then**
115: StoreBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id);
116: **end if**
117: Send-Sim(Receiver, "Detach Volume", session-id, volume-id, node-id);
118:
119: Upon receiving ("Result Detach Volume", session-id) from Sim:
120: **if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid) **then**
121: **if** valid=True AND attached-volume for node node-id is volume-id AND attached-node for volume volume-id is node-id **then**
122: Set attached-volume for node node-id to None
123: Set attached-node for volume volume-id to None
124: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid) }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success })
125: Send-Sim("Detach Volume", session-id, volume-id, node-id, success)
126: **else**
127: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid) }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure) })
128: Send-Sim("Detach Volume", session-id, volume-id, node-id, failure)
129: **end if**
130: **end if**
131:
132: Upon receiving ("Output Detach Volume", session-id) from Sim:
133: **if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success) **then**
134: RemoveBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, success)
135: Output("Detach Volume", session-id, success) to Receiver
136: **else if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure) **then**
137: RemoveBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, failure)
138: Output("Detach Volume", session-id, failure) to Receiver
139: **end if**
140:
141: Upon Receiving (Confirm, "Detach Volume" session-id, valid);

142: **if** FindBuffer(Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id) **then**
143: **if** user user-id is allowed to detach volume-id from node-id session session-id **then**
144: valid=True
145: **else**
146: valid=False
147: **end if**
148: **if** Corrupt[Identity]=Active **then**
149: valid = value sent by Sim
150: **end if**
151: UpdateBuffer({Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id }, {Receiver, "Detach Volume", session-id, *creds*, volume-id, node-id, valid })
152: Send-Sim(Receiver, "Detach Volume", session-id, volume-id, node-id, valid)
153: **end if**
154:
155: Upon receiving ("Detach Volume", session-id, volume-id, node-id) from Sim;
156: **if** Corrupt[Compute]=Active OR Corrupt[Node node-id ]=Active **then**
157: Set attached-volume for node node-id to None
158: **end if**
159: **if** Corrupt[Storage]=Active OR Corrupt[Volume volume-id ]=Active **then**
160: Set attached-node for volume volume-id to None
161: **end if**
162: Send-Sim ("Detach Volume", session-id, volume-id, node-id);
163:
164: $C_1-$ Upon receiving (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure, ServiceObject-List) from U:
165: **if** Corrupt[Compute]!="Active" **then**
166: **if** session-id =(U,N) where N non-repetitive for user U **then**
167: StoreBuffer (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure);
168: **end if**
169: **end if**
170: Send-Sim (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure, ServiceObject-List);
171:
172: $C_3I_{12}-$ Upon receiving ("Create Node Output", session-id) from Sim:
173: **if** FindBuffer(Receiver, "Create Node Continue", session-id, node-id, valid) **then**
174: Output("Node Created", session-id, node-id, valid) to *Receiver*;
175: **end if**
176:
177: $D_1-$ Upon receiving (Receiver, "Delete Node", session-id, node-id, user-id) from user U:
178: **if** Corrupt[Compute]!="Active" **then**
179: **if** session-id =(U,N) where N non-repetitive for user U **then**

```
180:            StoreBuffer (Receiver, "Delete Node", session-id, user-id, node-id);
181:        end if
182:    end if
183:    Send-Sim(Receiver, "Delete Node", session-id, user-id, node-id);
184:
185:    AC_2− Upon receiving ("MainComputeCorrupt", request) from Sim:
186:    if Corrupt[Compute]="Active" then
187:        Apply the request;
188:        Send-Sim("MainComputeCorruptResult",result);
189:    end if
190:
191:    D_3O_2− Upon receiving (Receiver, "Delete Node", session-id, node-id, valid) from Sim:
192:    if Corrupt[Compute]="Active" or Corrupt[Identity]="Active" then
193:        Output("Delete Node", session-id, node-id, valid) to Receiver;
194:    end if
195:
196:    D_3ID_4− Upon receiving ("Delete Node", session-id, node-id, valid) from Sim:
197:    if FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id) & Corrupt[Identity]="Active" then
198:        if valid=1 then
199:            if node node-id exists then
200:                NodeExist=1;
201:                Delete node with node-id;
202:            else
203:                NodeExist=0
204:            end if
205:            RemoveBuffer(Receiver, "Delete Node", session-id, user-id, node-id)
206:            Send-Sim (Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist);
207:        end if
208:    end if
209:
210:    Upon receiving ("Confirm", session-id) from Sim:
211:    if FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) then
212:        if user-id is valid & user-id is allowed to create node using Node-Structure for this session-id then
213:            if Corrupt[Image]="Active" then
214:                UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure})
215:                C_3I_1− Send-Sim (Receiver, "Create Node", session-id, user-id, "Corrupted Get Image");
216:            else
217:                if user-id is allowed to create node with image-id with Node-Structure & there exists image-id then
218:                    Corresponding-Image=get the image corresponding to image-id;
```

219: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image});
220: $C_4I-$ Send-Sim(Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image); % GOTO $C_5I$
221: **else**
222: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node Continue", session-id, NULL, Fail});
223: $C_4II-$ Send-Sim (Receiver, "Create Node", session-id, user-id, Fail, reason="Get Image", Node-Structure);% GOTO $C_5II$
224: **end if**
225: **end if**
226: **else**
227: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure},{Receiver, "Create Node Continue", session-id, NULL, Fail});
228: $C_4III-$ Send-Sim(Receiver, "Create Node", session-id, user-id, Fail, reason="Create Node", Node-Structure);% GOTO $C_5III$
229: **end if**
230: **else if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id) **then**
231: **if** user-id is valid & user-id is allowed to delete node node-id **then**
232: Valid=1;
233: **else**
234: Valid=0;
235: **end if**
236: UpdateBuffer ({Receiver, "Delete Node", session-id, user-id, node-id },{Receiver, "Delete Node", session-id, user-id, node-id, Valid});
237: $D_4-$ Send-Sim (Receiver, "Delete Node", session-id, user-id, node-id, Valid);
238: **end if**
239:
240: $C_3C_1O_2-$ Upon receiving (Receiver, "Create Node result", session-id, node-id, valid) from Sim:
241: **if** Corrupt[Compute]="Active" **then**
242: Output ("Node Created", session-id, node-id, valid) to *Receiver*;
243: **end if**
244:
245: $C_3I_9-$ Upon receiving (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image",Corresponding-Image, node-id) from Sim:
246: **if** Corrupt[Image]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure) **then**
247: **if** node-id is not used **then**
248: UpdateBuffer({Receiver, "Create Node", session-id, user-id, "Corrupted Get Image", Node-Structure}, {Receiver, "Create Node Continue", session-id, node-id, Successful});
249: Add (node-id, user-id, Corresponding-Image, Node-Structure) to the list of active nodes;
250: Send-Sim("Create Node Continue", session-id, user-id, successful); % GOTO $C_3I_{10}$

```
        else
            UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure,, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Fail});
            Send-Sim("Create Node Continue", session-id, user-id, Fail); % GOTO C_3 I_10
        end if
    end if

    C_3 C_1 I_2 − Upon receiving ("Get Image", session-id, user-id, image-id) from Sim:
    if Corrupt[Compute]="Active" then
        if user-id is valid & user-id is allowed to apply Request session-id then
            valid=1;
            if image image-id exists then
                Corresponding-Image= extract corresponding image of image-id;
            else
                Corresponding-Image=NULL;
            end if
        else
            valid=0;
            Corresponding-Image=NULL;
        end if
        Send-Sim("Get Image", session-id, user-id, image-id, Corresponding-Image, valid); % GOTO C_3 C_1 I_3
    end if

    C_3 ID_8 − Upon receiving ("Get Image", session-idimage-id) from Sim:
    if Corrupt[Identity]="Active" then
        if image image-id exists then
            Corresponding-Image= extract corresponding image of image-id;
        else
            Corresponding-Image=NULL;
        end if
        Send-Sim("Get Image", session-id, image-id, Corresponding-Image);
    end if

    C_3 ID_{12} − Upon receiving ("Node ID", session-id, node-id, Corresponding-Image) from Sim:
    if Corrupt[Identity]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) then
        if node-id is not used then
            Add( node-id, user-id, Node-Structure, Corresponding-Image) to the list of active nodes;
            UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure}, {Receiver, "Create Node Continue, session-id, node-id, Successful});
            Send-Sim("Create Node Continue", session-id, user-id, Successful);% GOTO C_3 I_10
        else
```

290: UpdateBuffer({Receiver, "Create Node", session-id, user-id, image-id, Node-Structure}, {Receiver, "Create Node Continue, session-id, node-id, Fail});
291: Send-Sim("Create Node Continue", session-id, user-id, Fail);%GOTO $C_3I_{10}$
292: **end if**
293: **end if**
294: $C_3ID_5^*-$ Upon receiving ("Create Node Output valid", session-id, valid) from Sim:
295: **if** Corrupt[Identity]="Active" & FindBuffer(Receiver, "Create Node", session-id, user-id, image-id, Node-Structure) **then**
296: **if** valid=0 **then**
297: Output("Create Node", session-id, NULL, Fail) to *Receiver*;
298: **end if**
299: **end if**
300:
301: $D_8-$ Upon receiving ("Delete Node", session-id, Continue) from Sim:
302: **if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id, Valid) **then**
303: **if** there is node with node-id **then**
304: NodeExist=1;
305: **else**
306: NodeExist=0;
307: **end if**
308: UpdateBuffer({Receiver, "Delete Node", session-id, user-id, node-id, Valid}, {Receiver, "Delete Node", session-id, user-id, node-id, Valid, NodeExist});
309: **if** Valid=1 & NodeExist=1 **then**
310: Delete node-id from the Node list;
311: **end if**
312: Send-Sim ("Delete Node Completed", session-id, Valid, NodeExist);
313: **end if**
314:
315: $D_{11}-$ Upon receiving ("Output Delete Node", session-id) from Sim:
316: **if** FindBuffer(Receiver, "Delete Node", session-id, user-id, node-id, Valid, NodeExist) **then**
317: **if** NodeExist=0 or Valid=0 **then**
318: Output("Delete Node", session-id, node-id, Fail) to *Receiver*;
319: **else**
320: Output("Delete Node", session-id, node-id, Successful) to *Receiver*;
321: **end if**
322: **end if**
323: $C_{13}I-$ Upon receiving ("Node ID", session-id, node-id) from Sim:
324: **if** FindBuffer(Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image) **then**
325: node-struct=Node-Structure;
326: **if** node-id is not used **then**
327: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Successful});
328: Add (node-id, user-id, Corresponding-Image, Node-Structure) to the list of active nodes;

329: Send-Sim("Create Node Continue", session-id, user-id, successful);
330: **else**
331: UpdateBuffer({Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure,, Corresponding-Image}, {Receiver, "Create Node Continue", session-id, node-id, Fail});
332: Send-Sim("Create Node Continue", session-id, user-id, Fail);
333: **end if**
334: **end if**
335:
336: $C_9III, C_{13}II, C_{16}I-$ Upon receiving ("Create Node Output", session-id) from Sim:
337: **if** FindBuffer(Receiver, "Create Node Continue", session-id, node-id, valid) **then**
338: Output("Create Node", session-id, node-id, valid) to *Receiver*;
339: **end if**
340: **end function**

## 6.2 OpenStack Services Functionalities Using One-Time Token

In the case of using One-time token, only Algorithm 10 and 25 are slightly changed. The new dashboard algorithm (Algorithm 30) creates one-time token credential per each request and adds it to the message before forwarding to the destination service.

The new identity algorithm (Algorithm 31) needs to make sure that the request is in the scope of the one-time token. It then adds the service (that made the validation request) to the session-id blacklist until the expiration of the one-time token.

**Algorithm 30** Dashboard for one-time token

1: **function** DASHBOARD
2:
3: Upon receiving Request-Message from Env:% The request-message also contains the expiration date and other information that are needed for the one-time Token
4: Creates one-time token *creds* for the specific request with time and resource limitation based on the Request-Message, MasterToken and Key;
5: Request-Message=Remove extra information that were needed to create one-time token from Request-Message;
6: User U Creates ($U_{Dashboard}$, $F_{Service}$, Request-Message, *creds*);
7: Send-$\mathcal{F}_{ExtNet}$($Dashboard_U$, $F_{Service}$, Request-Message, *creds*);
8:
9: Upon receiving (Sender, Output-Message) from $\mathcal{F}_{ExtNet}$:
10: Output (Output-Message) to Env;
11: **end function**

## 6.3 Simulator for Ideal Cloud with Improved Tokens

The simulator $S$ for the ideal cloud using one-time token mechanism is given in Algorithm 32 [6]. This simulator is more stringent than the simulator using bearer token model (Algorithm 26).

[6]The description of notation used in the algorithm is provided in appendix A

**Algorithm 31** Identity for one-time token

**function** CORRUPT
    Upon receiving ("Corrupt", Corrupt-Code) from Adv:
    Replace code of $\mathcal{F}_{Identity}$ with code of Corrupted $\mathcal{F}_{Identity}$
**end function**
**function** SERVICEVALIDATION % $F_{Identity}$ removes the black list of the expired tokens;
    Upon receiving (Receiver, "Service Validation", session-id, $creds_{service}$, *creds*, Request) from $\mathcal{F}_{SMT}$:
    **if** $creds_{service}$ & *creds* are valid & *creds* is allowed to perform the Request **then** % here the validation means: 1)It computes the HMAC related to that user and then compares 2) The token is not expired 3) this service needs to take action but is not in $BlackList_{\text{session-id}}$t.
        Add the service to $BlackList_{\text{session-id}}$;
        valid=1;
    **else**
        valid=0;
    **end if**
    Send-$\mathcal{F}_{SMT}$ ($\mathcal{F}_{Identity}$, *Receiver*, "Service Validated", session-id, user-id, service-id, Request, valid);
**end function**

To enforce the single-use constraint, $S$ creates a black list to store the services that have already checked the validity of a credential using session-id; the simulator then rejects all future attempts to validate the same credential. To enforce the scoping constraint, $S$ only issues credentials when given both an session-id *and* a request scope; the simulator records this intended scope and subsequently rejects all attempts to validate the credential in pursuit of a request outside of this scope.

## 6.4 Analysis

In this section, we explain how the OpenStack Services collectively UC-realize our Ideal Cloud in the one-time token scenario. Note that properties 1 and 2 of Section 4.4 continue to hold in the one-time token case because those properties only demonstrate correctness when all services are uncompromised, in which case there are no tokens (of any type) for the adversary to abuse. To complete the proof, we require two new properties.

- **Property 3'**: The view of $E$ is identical in the less-leaky $\mathcal{F}_{Cloud}$ and the OpenStack services even when services are compromised, as long as each observed token can only be used one time.
- **Property 4**: The cryptographic design of our stronger tokens ensures that each token may only be used once.

### 6.4.1 Property 3': Service compromise using one-time token model

The one-time token has two properties: 1- Scoped and 2- One-time usage. In the OpenStack Services, because of the first property, the adversary is only able to make requests in the defined scope and because of the second property, the adversary is not able to reuse it to make a request to

**Algorithm 32** Simulator with corrupted services for One-time token

1: **function** SIMULATOR(message)
2:
3: Upon receiving ("Corrupt", Service,value) from Env:
4: Corrupt[Service]=value;%value=passive or Active. For passive corruption, there would be no changes in the ideal cloud functionality. In this case, the credentials are leaked (which has not been written in the code.)
5: Fetch Buffer and find all session-id which are not finalized when corruption happens and create unique Credential per each, then update BufferRequest with the new credential and set corruption value to "corrupt";
6: **if** FindServiceCorrupt(Service, Service-id, $creds_{service}$) & $creds_{service}$ =NULL **then**
7: Credential= Generate credential for the service;
8: $creds_{service}$ =Credential;
9: UpdateServiceCorrupt(Service, Service-id, $creds_{service}$);
10: **end if**
11: Send-IdealCloud("Corrupt", Service, value);
12: Upon receiving (DB of (image-id, corresponding-Image) from Ideal Cloud:
13: StoreDB (All (image-id, corresponding-Image));
14:
15: Upon receiving (List of active nodes) from Ideal Cloud:
16: StoreDB (active nodes);
17: Upon receiving (Sender, Receiver, message) from Env:
18: **if** Receiver=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" **then**
19: Send-Env(Sender, Adversary, message);
20: **else if** Receiver=$\mathcal{F}_{Identity}$ & Corrupt[Identity]="Active" **then**
21: Send-Env(Sender, Adversary, message);
22: **else if** Receiver=$\mathcal{F}_{Image}$ & Corrupt[Image]="Active" **then**
23: Send-Env(Sender, Adversary, message);
24: **else if** Receiver=$\mathcal{F}_{BlockStorage}$ & Corrupt[Storage]="Active" **then**
25: Send-Env(Sender, Adversary, message);
26: **else if** Receiver=$\mathcal{F}_{Node}$node-id & Corrupt[Node node-id ]="Active" **then**
27: Send-Env(Sender, Adversary, message);
28: **else if** Receiver=$\mathcal{F}_{Volume}$volume-id & Corrupt[Volume volume-id ]="Active" **then**
29: Send-Env(Sender, Adversary, message);
30: **end if**

31:

32: Upon receiving (Receiver, "Attach Volume", session-id, user-id, node-id) from Cloud:

33: StoreBuffer(Receiver, "Attach Volume", session-id, user-id, node-id)

34: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, user-id, node-id)

35:

36: Upon Receiving("Attach Volume", session-id, volume-id, node-id, success)

37: StoreBuffer("Attach Volume", session-id, volume-id, node-id, success)

38: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, volume-id, node-id, success)

39:

40: Upon Receiving("Attach Volume", session-id, volume-id, node-id, failure)

41: StoreBuffer("Attach Volume", session-id, volume-id, node-id, failure)

42: Send-Env ($\mathcal{F}_{Compute}$, "Attach Volume", session-id, volume-id, node-id, failure)

43:

44: Upon Receiving("Attach Volume", session-id, volume-id, node-id, valid)

45: StoreBuffer("Attach Volume", session-id, volume-id, node-id, valid)

46: Send-Env ($\mathcal{F}_{Identity}$, "Attach Volume", session-id, volume-id, node-id, valid)

47:

48: Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Attach Volume", session-id, node-id, RequestActivity) from Env:

49: Send-IdealCloud(Adversary, "Attach Volume", session-id, node-id, RequestActivity)

50:

51: Upon Receiving ("Attach Volume", session-id, volume-id, node-id)

52: **if** (Compute is Corrupt OR Node node-id is Corrupt) AND (Storage is Corrupt OR Volume volume-id is Corrupt) **then**

53: Send-Env("Attach Volume", session-id, volume-id, node-id, success)

54: **else**

55: Send-Env("Attach Volume", session-id, volume-id, node-id, failure)

56: **end if**

57:

58: Upon receiving (Receiver, "Detach Volume", session-id, user-id, node-id) from Cloud:

59: StoreBuffer(Receiver, "Detach Volume", session-id, user-id, node-id)

60: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, user-id, node-id)

61:

62: Upon Receiving("Detach Volume", session-id, volume-id, node-id, success)

63: StoreBuffer("Detach Volume", session-id, volume-id, node-id, success)

64: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, volume-id, node-id, success)

65:

66: Upon Receiving("Detach Volume", session-id, volume-id, node-id, failure)

67: StoreBuffer("Detach Volume", session-id, volume-id, node-id, failure)

68: Send-Env ($\mathcal{F}_{Compute}$, "Detach Volume", session-id, volume-id, node-id, failure)

69:

70: Upon Receiving("Detach Volume", session-id, volume-id, node-id, valid)

71: StoreBuffer("Detach Volume", session-id, volume-id, node-id, valid)

72: Send-Env ($\mathcal{F}_{Identity}$, "Detach Volume", session-id, volume-id, node-id, valid)

73:

74: Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Detach Volume", session-id, node-id, RequestActivity) from Env:

75: Send-IdealCloud(Adversary, "Detach Volume", session-id, node-id, RequestActivity)

76:

77: Upon Receiving ("Detach Volume", session-id, volume-id, node-id)

78: **if** (Compute is Corrupt OR Node node-id is Corrupt) OR (Storage is Corrupt OR Volume volume-id is Corrupt) **then**

79: Send-Env("Detach Volume", session-id, volume-id, node-id, success)

80: **else**

81: Send-Env("Detach Volume", session-id, volume-id, node-id, failure)

82: **end if**

83:

84: $C_2-$ Upon receiving (Receiver, "Create Node", session-id, user-id, image-id, Node-Structure, ServiceObject-List) from Ideal cloud:

85: StoreDBRequest(session-id, Request-Message, NULL, ServiceObject-List, NULL, NULL);

86: StoreBuffer(Receiver,"Create Node", session-id, user-id, image-id, Node-Structure);

87: Send-Env(Receiver, $\mathcal{F}_{Compute}$, "Create Node", session-id, user-id, image-id, Node-Structure, ServiceObject-List);

88:

89: $AC_1-$ Upon receiving (Adversary, $\mathcal{F}_{Compute}$, request) from Env:
90: **if** Corrupt[Compute]="Active" **then**
91: Send-IdealCloud("MainComputeCorrupt", request);
92: **end if**
93:
94: $C_3C_1I_3-$ Upon receiving ("Get Image", session-id, Corresponding-Image, valid) from IdealCloud:
95: **if** FindBuffer("Result", Service, destination, message) with request-ID=session-id **then**
96: UpdateBuffer({"Result", $F_{Service}$, destination, message}, {$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid});
97: StoreDB(image-id, Corresponding-Image);% if this image is not stored in the Sim DB, then store it.
98: Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, FImage-id, user-id, "Get Image image-id ");
99: **end if**
100:
101: $C_3I_2-$ Upon receiving (Receiver, "Create Node", session-id, user-id, "Corrupted Get Image") from Ideal Cloud:
102: StoreBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image");
103: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node");
104:
105: $C_3ID_9-$ Upon receiving ("Get Image", session-id, image-id, Corresponding-Image) from Ideal Cloud:
106: **if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid) **then**
107: **if** Corresponding-Image=NULL **then**
108: valid=0;
109: **else**
110: valid=1;
111: **end if**
112: UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid},{Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid});
113: Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-id, image-id, , Corresponding-Image);
114: **end if**

115:

116: $C_3I_{10}-$ Upon receiving ("Create Node Continue", session-id, user-id, valid) from Ideal-Cloud:

117: **if** FindBuffer(Receiver, "Node ID continue", session-id, node-id) **then**

118: UpdateBuffer({Receiver, "Node ID continue", session-id },{Receiver, "Create Node Output", session-id })

119: Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);%GOTO $C_3I_{11}$

120: **else if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id) **then**

121: UpdateBuffer ({Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id },{Receiver, "Create Node Output", session-id });

122: Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);%GOTO $C_3I_{11}$

123: **end if**

124:

125: $C_{12}I, C_3I_8, C_3ID_{11}-$ Upon receiving (Adversary, $\mathcal{F}_{Compute}$, "Node ID", session-id, node-id) from Env:

126: **if** Corrupt[Image]="Active" & FindBuffer (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image) **then**

127: UpdateBuffer({Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image},{Receiver, "Node ID continue", session-id, node-id });

128: $C_3I_8-$ Send-IdealCloud (Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image, node-id);

129: **else if** Corrupt[Identity]="Active" & FindBuffer(Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid) **then**

130: UpdateBuffer({Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid},{Receiver,"Create Node node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid,node-id })

131: $C_3ID_{11}-$ Send-IdealCloud("Node ID", session-id, node-id, Corresponding-Image);

132: **else if** FindBuffer(Receiver, "Request NodeID", session-id) **then**

133: UpdateBuffer ({Receiver, "Request NodeID", session-id },{Receiver, "Node ID continue", session-id, node-id })

134: Send-IdealCloud ("Node ID", session-id, node-id);

135: **end if**

136:

137: Upon receiving (Adversary, Receiver, "Forward", message, destination) from Env:

138: **if** Receiver=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" **then**

139: **if** Destination is not a service **then**

140: StoreBuffer("Output", $\mathcal{F}_{Compute}$, destination, message);

141: **else**

142: StoreBuffer($\mathcal{F}_{Compute}$, destination, message);

143: **end if**

144: $C_3C_1-$ Send-Env($\mathcal{F}_{Compute}$, destination, message);% FOR OUTPUT GOTO $C_3C_1O_1$, ELSE GOTO $C_3C_1ID_1$ [identity] OR $C_3C_1I_1$ [glance]
145: **else if** Receiver=$\mathcal{F}_{Identity}$ & Corrupt[Identity]="Active" **then**
146: **if** destination=$\mathcal{F}_{Compute}$ & Request="Delete Node" & message is "Service Validated" type & FindBuffer(Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id) **then**
147: extract the valid value from the message;
148: UpdateBuffer({Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid});
149: $D_3ID_2-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);
150: **else if** destination=$\mathcal{F}_{Compute}$ & Request="Create Node with Node-Structure" & message is "Service Validated" type & FindBuffer(Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure) **then**
151: extract the valid value from the message;
152: UpdateBuffer({Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure}, {Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid});
153: $C_3ID_2-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);
154: **else if** destination=$\mathcal{F}_{Image}$ & Request="Get Image image-id " & message is "Service Validated" type & FindBuffer(Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid) **then**
155: extract the Image-valid value from the message;
156: UpdateBuffer({Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid}, {Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid});
157: $C_3ID_6-$ Send-Env($\mathcal{F}_{Identity}$, destination, message);
158: **else**
159: StoreBuffer($\mathcal{F}_{Identity}$, destination, message);
160: Send-Env($\mathcal{F}_{Identity}$, destination, message);
161: **end if**
162: **else if** Receiver=$\mathcal{F}_{Image}$ & Corrupt[Image]="Active" **then**
163: **if** destination= $\mathcal{F}_{Compute}$ & message=("Image", session-id, image-id, Corresponding-Image ) & FindBuffer(Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image") where the message session-ID=session-id **then**
164: UpdateBuffer ({Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image});
165: **else**
166: StoreBuffer($\mathcal{F}_{Image}$, destination, message);
167: **end if**
168: $C_3I_6-$ Send-Env($\mathcal{F}_{Image}$, destination, message);% GOTO $C_3I_7ID_1$ [identity] OR $C_3I_7$ [Compute])
169: **end if**

170:

171: $C_3I_6^*-$ Upon receiving (Adversary, $\mathcal{F}_{Image}$, "Get Image", image-id) from Env:

172: **if** Corrupt[Image]="Active" **then**

173: **if** image-id exist **then**

174: Corresponding-Image= extract the corresponding image for image-id;

175: **else**

176: Corresponding-Image= NULL;

177: **end if**

178: Send-Env ($\mathcal{F}_{Image}$, Adversary, "Image", image-id, Corresponding-Image );

179: **end if**

180:

181: $D_2-$ Upon receiving (Receiver, "Delete Node", session-id, user-id, node-id) from Ideal cloud:

182: StoreDBRequest(session-id, Request-Message, NULL, ServiceObject-List, NULL, NULL);

183: StoreBuffer(Receiver,"Delete Node", session-id, user-id, node-id);

184: Send-Env(Receiver, $\mathcal{F}_{Compute}$, "Delete Node", session-id, node-id, user-id);

185:

186: $AC_3-$ Upon receiving ("MainComputeCorruptResult", result) from IdealCloud:

187: Send-Env ($\mathcal{F}_{Compute}$, Adversary, result); % GOTO $C_3C_1$ (for forwarding), GOTO $AC_1$ (for Adv to make new request)

188:

189: $D_3ID_5-$ Upon receiving (Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist) from Ideal Cloud:

190: StoreBuffer(Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, Node-Exist);

191: **if** valid=1 & NodeExist=1 **then**

192: Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, successful );

193: **else**

194: Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, Fail );

195: **end if**

196:

197: Upon receiving ("Confirm", session-id) from Env:

198: **if** FindBuffer(Receiver,"Create Node", session-id, user-id, image-id, Node-Structure) **then**

199: **if** Corrupt[Compute]="Active" **then**

200: **if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**

201: *creds* =Credential, ServiceObject-List;

202: **else**

203: Credential= Generate credential for the user;

204: *creds* = Credential, ServiceObject-List;

205: UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";

206: **end if**

$C_3C-$ Send-Env(Receiver, Adversary, "Create Node", session-id, *creds*, image-id, Node-Structure);% GOTO $AC_1$ (Adv makes a new request), $C_3C_1$ (for forwarding)

**else if** Corrupt[Identity]="Active" **then**

UpdateBuffer({Receiver,"Create Node", session-id, user-id, image-id, Node-Structure},{Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure});

Request="Create Node with Node-Structure";

$C_3ID-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Service-id, user-id, Request, ServiceObject-List);

**else if** Corrupt[Image]="Active" **then**

$C_3I-$ Send-IdealCloud("Confirm", session-id);

**else**

$C_3-$ Send-IdealCloud("Confirm", session-id);

**end if**

**else if** FindBuffer(Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure) **then**

**if** Corrupt[Identity]="Active" **then**

**if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**

*creds* =Credential, ServiceObject-List;

**else**

Credential= Generate credential for the user;

*creds* = Credential, ServiceObject-List;

UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";

**end if**

Extract $creds_{service}$ from ServiceCorrupt(Service, Service-id, $creds_{service}$) where Service=$\mathcal{F}_{Compute}$;

UpdateBuffer({Receiver,"Create Node validation", session-id, user-id, image-id, Node-Structure},{Receiver,"Create Node validated", session-id, user-id, image-id, Node-Structure});

$C_3ID_1-$ Send-Env ($\mathcal{F}_{Compute}$, Adversary, "Service validation, session-id, $creds_{service}$, *creds*, Request); % GOTO $D_3ID_2$ OR ADV MAY SEND ARBITRARY MESSAGE TO A SERVICE

**end if**

**else if** FindBuffer(Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid) **then**

**if** Corrupt[Identity]="Active" **then**

**if** valid=0 **then**

UpdateBuffer({Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, valid});

$C_3ID_3^*-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$

**else**

UpdateBuffer({Receiver,"Create Node result", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid});

237: $C_3ID_3-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id);%GOTO $C_3ID_4$
238: **end if**
239: **end if**
240: **else if** FindBuffer(Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, valid) **then**
241: $C_3ID_4^*-$ Send-IdealCloud("Create Node Output valid", session-id, Fail);
242: **else if** FindBuffer(Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid) **then**
243: UpdateBuffer({Receiver,"Create Node Get Image", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid});
244: $C_3ID_4-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, compute-id, user-id, Request="Get Image image-id ");
245: **else if** FindBuffer(Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid) **then**
246: **if** Corrupt[Identity]="Active" **then**
247: **if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**
248: *creds* =Credential, ServiceObject-List;
249: **else**
250: Credential= Generate credential for the user;
251: *creds* = Credential, ServiceObject-List;
252: UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";
253: **end if**
254: Extract $creds_{service}$ from ServiceCorrupt(Service, Service-id, $creds_{service}$) where Service=$\mathcal{F}_{Image}$;
255: UpdateBuffer({Receiver,"Create Node Get Image validation", session-id, user-id, image-id, Node-Structure, valid},{Receiver,"Create Node Get Image validated", session-id, user-id, image-id, Node-Structure, valid});
256: $C_3ID_5-$ Send-Env ($\mathcal{F}_{Image}$, Adversary, "Service validation, session-id, $creds_{service}$, *creds*, Request="Get Image image-id ");
257: **end if**
258: **else if** FindBuffer(Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid) **then**
259: **if** Corrupt[Identity]="Active" **then**
260: **if** Image-valid=0 **then**
261: UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid}, {Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, Fail});
262: $C_3ID_7-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$
263: **else**
264: UpdateBuffer({Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid},{Receiver,"Create Node Get Image result", session-id, user-id, image-id, Node-Structure, Image-valid});

$C_3ID_7-$ Send-IdealCloud("Get Image", session-id, image-id);%GOTO $C_3ID_8$
**end if**
**end if**
**else if** FindBuffer(Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid) **then**
**if** Corrupt[Identity]="Active" & valid=0 **then**
UpdateBuffer({Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid}, {Receiver,"Create Node result continue", session-id, user-id, image-id, Node-Structure, Fail});
$C_3ID_{10}-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Create Node", session-id, NULL, Fail);%GOTO $C_3ID_4^*$
**else if** Corrupt[Identity]="Active" & valid=1 **then**
UpdateBuffer({Receiver,"Create Node Get Image continue", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid}, {Receiver,"Create Node get node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image, valid});
$C_3ID_{10}-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);%GOTO $C_3ID_{11}$
**end if**
**else if** FindBuffer($\mathcal{F}_{Compute}$, destination, message) where the request-ID in the message is equal to session-id **then**
**if** destination=$\mathcal{F}_{Identity}$ **then**
**if** Corrupt[Identity] !="Active" **then**
**if** FindDBRequest(session-id, Credential, ServiceObject-List, Black-List) where request-ID=session-id & Credential=*creds* & ($\mathcal{F}_{Compute}$,Object) $\in$ ServiceObject-List & $\mathcal{F}_{Compute} \notin$ Black-List **then**
AddBlack-List$_{\text{session-id}}$($\mathcal{F}_{Compute}$);
valid=successful;
**else**
valid=fail;
**end if**
UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message}, {Receiver=destination, Sender= $\mathcal{F}_{Compute}$, "Service validated", session-id, Request, valid});
$C_3C_1ID_1-$ Send-Env(Receiver, Sender, "Service validated", Request, valid); % GOTO $C_3C_1ID_2$
**else**
Send-Env($\mathcal{F}_{Compute}$, Adversary, message);% adversary can either forward a message or make a create/delete node request.
**end if**
**else if** destination=$\mathcal{F}_{Image}$ **then**
**if** Corrupt[Image] !="Active" **then**
**if** FindDBRequest(session-id, Credential, ServiceObject-List, Black-List) where request-ID=session-id & Credential=*creds* & ($\mathcal{F}_{Image}$,image-id) $\in$ ServiceObject-List & $\mathcal{F}_{Image} \notin$ Black-List **then**
extract user-id from the message;
Message= message that the user credential is replaced by user-id;
UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message},{"Result", $\mathcal{F}_{Compute}$, destination, message});

296: $C_3C_1I_1-$ Send-IdealCloud(Message); %GOTO $C_3C_1I_2$
297: **else**
298: UpdateBuffer({$\mathcal{F}_{Compute}$, destination, message},{$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, NULL, Fail});
299: $C_3C_1I_1-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, FImage-id, user-id, "Get Image image-id ");%GOTO $C_3C_1I_4$
300: **end if**
301: **else**
302: Send-Env($\mathcal{F}_{Compute}$, Adversary, message);% adversary can either forward a message or make a create/delete node request.
303: **end if**
304: **end if**
305: **else if** FindBuffer(Receiver, Sender, "Service validated", session-id, Request, valid) **then**
306: **if** Sender=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" **then**
307: $C_3C_1ID_2-$ Send-Env(Receiver, Adversary, message);
308: **else if** Sender=$\mathcal{F}_{Image}$ & Corrupt[Image]="Active" **then**
309: $C_3I_7ID_2-$ Send-Env(Receiver, Adversary, message);
310: **end if**
311: **else if** FindBuffer($\mathcal{F}_{Image}$, destination, message) where the session-ID in the message is equal to session-id **then**
312: **if** destination= $\mathcal{F}_{Compute}$ & message=("Image", session-id, image-id, Corresponding-Image ) & FindBuffer(Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image") where the message request-ID=session-id **then**
313: UpdateBuffer({Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Node ID", session-id, user-id, image-id, "Corrupted Get Image", Corresponding-Image});
314: RemoveBuffer($\mathcal{F}_{Image}$, destination, message);
315: $C_3I_7-$ Send-Env ($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);
316: **else if** destination=$\mathcal{F}_{Identity}$ **then**
317: **if** Corrupt[Identity] !="Active" **then**
318: **if** FindDBRequest(session-id, Credential, ServiceObject-List, Black-List) where request-ID=session-id & Credential=*creds* & ($\mathcal{F}_{Image}$,Object) $\in$ ServiceObject-List & $\mathcal{F}_{Image} \notin$ Black-List **then**
319: AddBlack-List$_{\text{session-id}}$($\mathcal{F}_{Image}$);
320: valid=successful;
321: **else**
322: valid=fail;
323: **end if**
324: StoreBuffer(Receiver=destination, Sender= $\mathcal{F}_{Image}$, "Service validated", session-id, Request, valid);
325: $C_3I_7ID_1-$ Send-Env(Receiver, Sender, "Service validated", Request, valid);
326: **else**
327: Send-Env($\mathcal{F}_{Image}$, Adversary, message);
328: **end if**
329: **end if**

330: **else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid") **then**
331: AddBlack-List$_{\text{session-id}}$($\mathcal{F}_{Image}$);
332: UpdateBuffer({$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Get Image", session-id, image-id, Corresponding-Image, valid"},{$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid"});
333: $C_3C_1I_4-$ Send-Env ($\mathcal{F}_{Identity}$, $\mathcal{F}_{Image}$, "Service Validated", session-id, user-id, service-id, Request="Get Image image-id ", valid);
334: **else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid") **then**
335: UpdateBuffer({$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "validated Get Image", session-id, image-id, Corresponding-Image, valid"}, {$\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid"});
336: $C_3C_1I_5-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-idimage-id, Corresponding-Image);
337: **else if** FindBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid") **then**
338: RemoveBuffer($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Result Get Image", session-id, image-id, Corresponding-Image, valid");
339: **if** Corrupt[Compute]="Active" **then**
340: $C_3C_1I_6-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, message);
341: **end if**
342: **else if** FindBuffer("Output", Service, destination, message) **then**
343: RemoveBuffer("Output", Service, destination, message);
344: **if** Service=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" & message contains "Delete Node" **then**
345: $D_3O_1-$ Send-IdealCloud(Receiver, "Delete Node", session-id, node-id, valid);
346: **else if** Service=$\mathcal{F}_{Compute}$ & Corrupt[Compute]="Active" & message contains "Create Node" **then**
347: $C_3C_1O_1-$ Send-IdealCloud(Receiver, "Create Node result", session-id, node-id, valid);
348: **end if**
349: **else if** FindBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image") **then**
350: AddBlack-List$_{\text{session-id}}$($\mathcal{F}_{Compute}$);
351: UpdateBuffer({Receiver, "Service Validation", session-id, user-id, image-id, "Corrupted Get Image"},{Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image"})
352: $C_3I_3-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, Compute-id, "Create Node", Successful);
353: **else if** FindBuffer(Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image") **then**
354: UpdateBuffer({Receiver, "Service Validatied", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image"})
355: $C_3I_4-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id);

356: **else if** FindBuffer(Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image") **then**
357: UpdateBuffer({Receiver, "Get Image", session-id, user-id, image-id, "Corrupted Get Image"}, {Receiver, "Get Image Result", session-id, user-id, image-id, "Corrupted Get Image"});
358: **if** Corrupt[Image]="Active" **then**
359: **if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**
360: *creds* =Credential, ServiceObject-List;
361: **else**
362: Credential= Generate credential for the user;
363: *creds* = Credential, ServiceObject-List;
364: UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";
365: **end if**
366: $C_3I_5-$ Send-Env($\mathcal{F}_{Image}$, Adversary, message = ("Get Image", session-id, *creds*, image-id));% GOTO $C_3I_6$ FOR FORWARDING, $C_3I_6^*$ FOR MAKEING NEW REQUEST
367: **end if**
368: **else if** FindBuffer(Receiver, "Create Node Output", session-id) **then**
369: RemoveBuffer(Receiver, "Create Node Output", session-id);
370: $C_3I_{11}-$ Send-IdealCloud("Create Node Output", session-id);
371:
372: **else if** FindBuffer(Receiver, "Delete Node Completed", session-id, user-id, node-id, valid, NodeExist) **then**
373: $D_3ID_6-$ Send-IdealCloud(Receiver, "Delete Node", session-id, node-id, valid);%GOTO $D_3O_2$
374: **else if** FindBuffer(Receiver,"Delete Node", session-id, user-id, node-id) **then**
375: **if** Corrupt[Compute]="Active" **then**
376: **if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**
377: *creds* =Credential, ServiceObject-List;
378: **else**
379: Credential= Generate credential for the user;
380: *creds* = Credential, ServiceObject-List;
381: UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";
382: **end if**
383: $D_3C-$ Send-Env(Receiver, Adversary, "Delete Node", session-id, node-id, *creds*);
384: **else if** Corrupt[Identity]="Active" **then**
385: UpdateBuffer({Receiver,"Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id });
386: $D_3ID-$ Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Delete Node");
387: **else**
388: $D_3-$ Send-IdealCloud("Confirm", session-id);
389: **end if**

390: **else if** FindBuffer(Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id) **then**
391: **if** Corrupt[Identity]="Active" **then**
392: **if** FindDBRequest(session-id, Request-Message, Credential, ServiceObject-List, Black-List, Corruption) & Corruption ="Corrupt" **then**
393: *creds* =Credential, ServiceObject-List;
394: **else**
395: Credential= Generate credential for the user;
396: *creds* = Credential, ServiceObject-List;
397: UpdateDBRequest for session-id with the new credential value and set Corruption="Corrupt";
398: **end if**
399: Extract $creds_{service}$ from ServiceCorrupt(Service, Service-id, $creds_{service}$) where Service=$\mathcal{F}_{Compute}$;
400: UpdateBuffer({Receiver,"Corrupt Validation Delete Node", session-id, user-id, node-id }, {Receiver,"Corrupt Validated Delete Node", session-id, user-id, node-id });
401: $D_3ID_1-$ Send-Env(Receiver, Adversary, "Service Validation", session-id, $creds_{service}$, *creds*);
402: **end if**
403: **else if** FindBuffer(Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid) **then**
404: **if** Corrupt[Identity]="Active" **then**
405: **if** valid=0 **then**
406: UpdateBuffer({Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid}, {Receiver,"Corrupt Output Delete Node", session-id, user-id, node-id, valid});
407: $D_3ID_3-$ Send-Env($\mathcal{F}_{Compute}$, Receiver, "Delete Node", session-id, node-id, Fail);%GOTO $D_3ID_4^*$
408: **else**
409: RemoveBuffer(Receiver,"Corrupt Result Delete Node", session-id, user-id, node-id, valid);
410: $D_3ID_3-$ Send-IdealCloud("Delete Node", session-id, node-id, valid); %GOTO $D_3ID_4$
411: **end if**
412: **end if**
413: **else if** FindBuffer(Receiver,"Corrupt Output Delete Node", session-id, user-id, node-id, valid) **then**
414: $D_3ID_4^*-$ Send-IdealCloud(Receiver,"Delete Node", session-id, node-id, valid);%GOTO $D_3O_2$
415: **else if** FindBuffer(Receiver, "Node Access", session-id, user-id, node-id, RequestActivity) **then**
416: Send-IdealCloud("Result Access Node", session-id)
417: **else if** FindBuffer(Receiver, "Node Access", session-id, user-id, node-id, result, valid) **then**
418: Send-IdealCloud("Output Access Node", session-id)
419: **else if** FindBuffer(Receiver, "Attach Volume", session-id, volume-id, node-id) **then**
420: **if** Identity is Corrupt **then**
421: valid = validity setting used by simulated adversary

**else**
valid=False
**end if**
Send-IdealCloud("Confirm Attach Volume", session-id, valid);
**else if** FindBuffer("Attach Volume", session-id, volume-id, node-id, valid) **then**
Send-IdealCloud("Result Attach Volume", session-id)
**else if** FindBuffer("Attach Volume", session-id, volume-id, node-id, success/fail) **then**
Send-IdealCloud("Output Attach Volume", session-id)
**else if** FindBuffer(Receiver, "Detach Volume", session-id, volume-id, node-id) **then**
**if** Identity is Corrupt **then**
valid = validity setting used by simulated adversary
**else**
valid=False
**end if**
Send-IdealCloud("Confirm Detach Volume", session-id, valid)
**else if** FindBuffer("Detach Volume", session-id, volume-id, node-id, valid) **then**
Send-IdealCloud("Result Detach Volume", session-id)
**else if** FindBuffer("Detach Volume", session-id, volume-id, node-id, success/fail) **then**
Send-IdealCloud("Output Detach Volume", session-id);
**else if** FindBuffer(Receiver, "Delete Service Validation", session-id, Service-id, user-id, node-id, valid) **then**
$D_6-$ Send-Env ($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, service-id, Request, valid);
**else if** FindBuffer(Receiver, "Delete Service Validated", session-id, user-id, node-id, valid) **then**
UpdateBuffer({Receiver, "Delete Service Validated", session-id, user-id, node-id, valid},{Receiver, "Delete Node Continue", session-id, user-id, node-id, valid});
$D_7-$ Send-IdealCloud ("Delete Node Output", session-id, continue);
**else if** FindBuffer(Receiver, "Output Delete Node", session-id) **then**
$D_{10}-$ Send-IdealCloud("Output Delete Node", session-id);
**else if** FindBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image) **then**
UpdateBuffer({Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Service Validated", session-id, user-id, image-id, Service-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image})
Add Black-List$_{\text{session-id}}$($\mathcal{F}_{Compute}$);
$C_6-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Compute}$, "Service Validated", session-id, user-id, Compute-id, "Create Node", create-valid);
**else if** FindBuffer(Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image) **then**
**if** create-valid=successful **then**

454: UpdateBuffer({Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image})

455: $C_7I, C_7II-$ Send-Env ($\mathcal{F}_{Compute}$, $\mathcal{F}_{Image}$, "Get Image", session-id, user-id, image-id, ServiceObject-List);

456: **else**

457: UpdateBuffer({Receiver, "Service Validated", session-id, user-id, image-id, Compute-id, "Create Node", create-valid, "Get Image", Image-valid, Corresponding-Image},{Receiver, "Create Node Output", session-id });

458: $C_7III-$ Send-Env($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, NULL, Fail);

459: **end if**

460: **else if** FindBuffer(Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image) **then**

461: UpdateBuffer({Receiver, "Get Image Create Node", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Service Validation", session-id, image-id, FImage-id, Image-valid, Corresponding-Image});

462: $C_8I, C_8II-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, FImage-id, user-id, "Get Image Image-id", ServiceObject-List);

463: **else if** FindBuffer(Receiver, "Get Image Service Validation", session-id, image-id, FImage-id, Image-valid, Corresponding-Image) **then**

464: UpdateBuffer({Receiver, "Get Image Service Validation", session-id, user-id, image-id, Compute-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image})

465: Add Black-List$_{\text{session-id}}$($\mathcal{F}_{Image}$);

466: $C_9I, C_9II-$ Send-Env($\mathcal{F}_{Identity}$, $\mathcal{F}_{Image}$, "Service Validated", session-id, user-id, Fimage-id, "Get Image image-id", Image-valid, ServiceObject-List);

467: **else if** FindBuffer(Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image) **then**

468: **if** Image-valid =false **then**

469: Corresponding-Image=NULL

470: **end if**

471: UpdateBuffer({Receiver, "Get Image Service Validated", session-id, image-id, FImage-id, Image-valid, Corresponding-Image},{Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Image})

472: $C_{10}I, C_{10}II-$ Send-Env($\mathcal{F}_{Image}$, $\mathcal{F}_{Compute}$, "Image", session-id, image-id, Corresponding-Image);

473:

474: **else if** FindBuffer(Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Image) **then**
475: **if** Corresponding-Image=NULL **then**
476: UpdateBuffer({Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Imaged},{Receiver, "Create Node Output", session-id })
477: $C_{11}II-$ Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, NULL, Fail);
478: **else**
479: UpdateBuffer({Receiver, "Get Image Continue", session-id, image-id, Image-valid, Corresponding-Imaged},{Receiver, "Request NodeID", session-id })
480:
481: $C_{11}I-$ Send-Env($\mathcal{F}_{Compute}$, Adversary, "Request Node ID", session-id);
482: **end if**
483:
484: **else if** FindBuffer(Receiver, "Create Node Output", session-id) **then**
485: RemoveBuffer (Receiver, "Create Node Output", session-id);
486: $C_8III, C_{12}II, C_{15}I-$ Send-IdealCloud ("Create Node Output", session-id);
487:
488: **end if**
489:
490: $D_5-$ Upon receiving (Receiver, "Delete Node", session-id, user-id, node-id, valid) from Ideal Cloud:
491: Service-id=Service-id for $F_{Compute}$
492: UpdateBuffer({Receiver,"Delete Node", session-id, user-id, node-id }, {Receiver, "Delete Service Validation", session-id, Service-id, user-id, node-id, valid});
493: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Delete node-id ");
494:
495: $D_9-$ Upon receiving( "Delete Node Completed", session-id, Valid, NodeExist) from Ideal-Cloud:
496: **if** FindBuffer(Receiver, "Delete Node Continue", session-id, user-id, node-id, valid) **then**
497: UpdateBuffer ({"Delete Node Continue", session-id, user-id, node-id, valid}, {"Output Delete Node", session-id })
498: **if** NodeExist=0 or Valid=0 **then**
499: Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Delete Node", session-id, node-id, Fail);
500: **else**
501: Send-Env ($\mathcal{F}_{Compute}$, *Receiver,* "Delete Node", session-id, node-id, Successful);
502: **end if**
503: **end if**

504: $C_5I-$ Upon receiving (Receiver, "Get Node ID", session-id, user-id, image-id, Node-Structure, Corresponding-Image) from Ideal Cloud:
505: StoreBuffer(Receiver, "Service Validation", session-id, user-id, image-id, "Create Node", Successful, "Get Image", Successful, Corresponding-Image);
506: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node Node-Structure with ServiceObject-List ");% GOTO $C_6$
507:
508: $C_5II-$ Upon receiving (Receiver, "Create Node", session-id, user-id, Fail, reason="Get Image", Node-Structure) from Ideal Cloud:
509: StoreBuffer(Receiver, "Service Validation", session-id, user-id, "Create Node", Successful, "Get Image", Fail, NULL);
510: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node Node-Structure with ServiceObject-List ");% GOTO $C_6$
511:
512: $C_5III-$ Upon receiving (Receiver, "Create Node", session-id, user-id, Fail, reason="Create Node", Node-Structure) from Ideal Cloud:
513: StoreBuffer(Receiver, "Service Validation", session-id, user-id, "Create Node", Fail, "Get Image", Fail, NULL);
514: Send-Env($\mathcal{F}_{Compute}$, $\mathcal{F}_{Identity}$, "Service Validation", session-id, Compute-id, user-id, "Create Node Node-Structure with ServiceObject-List " );% GOTO $C_6$
515:
516: $C_{14}I-$ Upon receiving ("Create Node Continue", session-id, user-id, valid) from IdealCloud:
517: **if** FindBuffer(Receiver, "Node ID continue", session-id, node-id) **then**
518: UpdateBuffer({Receiver, "Node ID continue", session-id },{Receiver, "Create Node Output", session-id })
519: Send-Env ($\mathcal{F}_{Compute}$, *Receiver*, "Node Created", session-id, node-id, valid);
520: **end if**
521: **end function**

the same service. To realize these properties, the simulator no longer store credentials per user-id; instead, it creates and stores one credential per each compromised session-id.

That is, when Nova is compromised, the simulator notifies the ideal cloud and the ideal cloud sends all session-id in which the requests are either new to Nova or not finalized. Next, the simulator creates credentials for each session-id, stores them together with the request detail (scope) in its database for future use, and adds these credentials to the corresponding messages as needed. In addition, because of second property, the simulator creates a blacklist for each session-id and adds the services which have made a request for validation. In this way, the simulator is able to track all service requests for validation and properly respond to the requests such that $E$ views identical messages from both worlds.

When an adversary uses a compromised session-id =5 with request detail of "CreateNode with image-id =2" and makes a GetImage request to Glance for image-id =2, the Glance needs to validate the credential first then send the corresponding image. For this case, the simulator creates all the intermediate messages until it reaches the validation result message to Glance. If Glance has previously made a request with same credential for session-id =5, then the simulator will send a failed result (because this service is in the blacklist of session-id =5 in the simulator). If the credential is correct, then $S$ first adds Glance to the blacklist of session-id =5 and then $S$ sends the successful result. This means that if the specification of the corresponding image related to image-id =2 changes in the future, then the adversary *cannot* access the more recent and up to date information which is what happens in the OpenStack Services.

Note that even though getting access to the most recent image specification may not be a great interest for the adversary, but the point here is that the adversary cannot get the updates for any other request in which the update information is crucial.

On the hand, because of one-time token's scope property, the simulator stores the request detail and does not permit a request with different scope for the same session-id. That is (in our example), if the adversary makes a request of accessing corresponding image of image-id =7 for session-id =5, then the simulator will send a fail result (because the scope of this request is "create a node with image-id =2 for session-id =5").

### 6.4.2 Property 4: Cryptographic design ensures one time use

In this section, we demonstrate that the Recursive Augmented Fernet Token (RAFT) described in Algorithm 28 suffices to ensure one-time use. Because the construction involves cryptography, this is the only part of the proof in which simulation only holds against computationally-bound adversaries.

The strength of RAFT boils down to the intersection of three properties:

1. The cryptographic strength of HMAC, which is unforgeable by any computationally-constrained adversary who does not possess the key.

2. The method by which keys are doled out by Keystone, and then recursively by users to services, to ensure that the HMAC can only be performed by authorized services.

3. $\mathcal{F}_{Identity}$'s validation algorithm, which sends an alert to an (uncompromised) service if any of the following hold: the token is reused, the HMAC is invalid, or a service produces an HMAC for a request that isn't within the authorized scope of a prior request in the chain.

We apply the three properties in sequence to realize the scoped and single-use properties of RAFT. First, by a standard hybrid argument it suffices to replace the pseudorandom HMAC with a truly random function. Via a reasoning up-to-bad argument, we may safely condition the remainder of our argument on the (overwhelming) likelihood that a forgery doesn't occur, in which case our invariant is maintained that $E$'s view is unchanged. The second property demonstrates that if forgeries do not occur, then the recursive tokens form a chain of requests that ultimately tie back to a single authorization made by the user. Finally, the third property demonstrates both that the token can only be used once and also that the token preserves scope; more specifically, the tokens in the chain 'link' together in the sense that the request made by each service must be a logical consequence of the request made by a prior service.

# 7 Conclusion & Future Work

This work lays the foundation for a full-scope composable security analysis for the popular cloud management framework OpenStack. We believe that this effort can bring communities together. On the one hand, our abstract model is substantially easier to absorb than the code of OpenStack and therefore opens OpenStack to a wider group within the cryptography and programming languages communities. On the other hand, our modular analysis provides the OpenStack development community with a better understanding of the security concerns surrounding some of its core design issues as well as (more importantly) a concise, tangible description of how the security of the overall cloud concretely improves by making moderate software improvements. Put differently: the modular analysis can both expose bugs and provide motivation for the developers to address them.

Even so, this work covers only some of the core functionality provided by a full featured cloud. In particular, there are still many remaining features necessary for fine grained access control. Much more work is needed in order to: (a) cover more services, (b) consider more attack (corruption) options and the security guarantees provided in these cases, and (c) analyze implementations of the various services and the associated security caveats and vulnerabilities. The last point is where the power of universal composability is put to use: our model can be extended to include internal components of services such as Compute and Storage in order to show how these subsystems combine to realize the ideal services $\mathcal{F}_{Compute}$ and $\mathcal{F}_{BlockStorage}$ that we use in this work. This level of modeling also enables clearer discussion of the effects of single-node compromises.

Another natural direction for future research is mechanizing the analysis, and in particular the proofs of security, such as the one done here. Indeed, we are already taking first steps toward formalizing UC security within the EasyCrypt computer-aided proof system.

We believe that this work provides an important benchmark to show the feasibility of modeling large-scale software packages within the framework of Universal Composability. Modularity was a crucial component toward keeping the models and analyses manageable. Indeed, this work demonstrates the value of conducting similar analyses of other software deployments in the future.

# References

[1] B. Pfitzmann and M. Waidner, "Composition and integrity preservation of secure reactive systems," 2000, pp. 245–254.

[2] M. Backes, B. Pfitzmann, and M. Waidner, "A composable cryptographic library with nested operations," in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, 2003, pp. 220–230.

[3] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145.

[4] M. Backes, J. Dreier, S. Kremer, and R. Künnemann, "A novel approach for reasoning about liveness in cryptographic protocols and its application to fair exchange." IEEE Computer Society, 2017.

[5] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 443–461.

[6] C. Kudla and K. G. Paterson, "Modular security proofs for key agreement protocols," in *Advances in Cryptology - ASIACRYPT 2005*, B. Roy, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 549–565.

[7] A. Duc, S. Dziembowski, and S. Faust, "Unifying leakage models: From probing attacks to noisy leakage." P. Q. Nguyen and E. Oswald, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 423–440.

[8] K. Bhargavan, B. Bond, A. Delignat-Lavaud, C. Fournet, C. Hawblitzel, C. Hritcu, S. Ishtiaq, M. Kohlweiss, R. Leino, J. R. Lorch, K. Maillard, J. Pan, B. Parno, J. Protzenko, T. Ramananandro, A. Rane, A. Rastogi, N. Swamy, L. Thompson, P. Wang, S. Z. Béguelin, and J. K. Zinzindohoue, "Everest: Towards a verified, drop-in replacement of HTTPS," in *2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA*, 2017, pp. 1:1–1:12.

[9] R. Canetti, S. Chari, S. Halevi, B. Pfitzmann, A. Roy, M. Steiner, and W. Venema, "Composable security analysis of OS services," in *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, 2011, pp. 431–448.

[10] R. Gu, J. Koenig, T. Ramananandro, Z. Shao, X. N. Wu, S. Weng, H. Zhang, and Y. Guo, "Deep specifications and certified abstraction layers," in *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. ACM, 2015, pp. 595–608.

[11] R. Gu, Z. Shao, H. Chen, X. N. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "Certikos: An extensible architecture for building certified concurrent OS kernels," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, 2016, pp. 653–669.

[12] H. Chen, X. N. Wu, Z. Shao, J. Lockerman, and R. Gu, "Toward compositional verification of interruptible OS kernels and device drivers," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, 2016, pp. 431–447.

[13] C. Metz, "The secret history of OpenStack, the free cloud software that's changing everything," *WIRED*, April 2012. [Online]. Available: https://www.wired.com/2012/04/openstack-3/

[14] A. Venkatraman, "OpenStack market size will cross $1.7bn by 2016, says 451 research," August 2014. [Online]. Available: http://www.computerweekly.com/news/2240226930/OpenStack-market-size-will-cross-17bn-by-2016-says-451-Research

[15] Heidi Joy Tretheway. (2017, April) Users stand up, speak out, and deliver data on OpenStack growth. [Online]. Available: https://opensource.com/article/17/4/openstack-user-survey

[16] O. G. Body, 2016. [Online]. Available: https://governance.openstack.org/

[17] OpenStack Security Group, "Openstack security guide," 2015.

[18] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Collaboration-based cloud computing security management framework," in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 364–371.

[19] L. M. Vaquero, L. Rodero-Merino, and D. Morán, "Locking the sky: a survey on iaas cloud security," *Computing*, vol. 91, no. 1, pp. 93–118, 2011. [Online]. Available: https://dx.doi.org/10.1007/s00607-010-0140-x

[20] L. Gu, A. Vaynberg, B. Ford, Z. Shao, and D. Costanzo, "Certikos: A certified kernel for secure cloud computing," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, ser. APSys '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:5. [Online]. Available: http://doi.acm.org/10.1145/2103799.2103803

[21] H. Albaroodi, S. Manickam, and P. Singh, "Critical review of openstack security: Issues and weaknesses," *Journal of Computer Science*, vol. 10, no. 1, pp. 23–33, 2014.

[22] R. Slipetskyy, "Security issues in openstack," *Master's thesis, Norwegian University of Science and Technology*, 2011.

[23] Ericsson. (2013) Keystone security gap and threat identification. [Online]. Available: https://wiki.openstack.org/w/images/c/c9/OpenStack_Keystone_Analysis.pdf

[24] W. K. Sze, A. Srivastava, and R. Sekar, "Hardening openstack cloud platforms against compute node compromises," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 341–352.

[25] P. Desnoyers, J. Hennessey, B. Holden, O. Krieger, L. Rudolph, and A. Young, "Using open stack for an open cloud exchange(OCX)," in *2015 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 48–53.

[26] R. Canetti, K. Hogan, A. Malhotra, and M. Varia, "A universally composable treatment of network time," in *30th IEEE Computer Security Foundations Symposium, CSF*. IEEE Computer Society, 2017, pp. 360–375. [Online]. Available: https://doi.org/10.1109/CSF.2017.38

[27] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub, "Easycrypt: A tutorial," in *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, ser. Lecture Notes in Computer Science, vol. 8604. Springer, 2013, pp. 146–166. [Online]. Available: https://doi.org/10.1007/978-3-319-10082-1_6

[28] A. Petcher and G. Morrisett, "The foundational cryptography framework," in *Principles of Security and Trust - 4th International Conference, POST*, ser. Lecture Notes in Computer Science, vol. 9036. Springer, 2015, pp. 53–72. [Online]. Available: https://doi.org/10.1007/978-3-662-46666-7_4

[29] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "Crypthol: Game-based proofs in higher-order logic," *IACR Cryptology ePrint Archive*, vol. 2017, p. 753, 2017. [Online]. Available: http://eprint.iacr.org/2017/753

[30] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono, "All your clouds are belong to us: Security analysis of cloud management interfaces," in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, ser. CCSW '11. New York, NY, USA: ACM, 2011, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2046660.2046664

[31] Y. Sun, G. Petracca, T. Jaeger, H. Vijayakumar, and J. Schiffman, "Cloud armor: Protecting cloud commands from compromised cloud services," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 253–260.

[32] Y. Sun, G. Petracca, and T. Jaeger, "Inevitable failure: The flawed trust assumption in the cloud," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*. ACM, 2014, pp. 141–150.

[33] Y. Sun, G. Petracca, X. Ge, and T. Jaeger, "Pileus: Protecting user resources from vulnerable cloud services," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 52–64.

[34] R. Canetti, S. Chari, S. Halevi, B. Pfitzmann, A. Roy, M. Steiner, and W. Venema, "Composable security analysis of OS services," in *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6715, 2011, pp. 431–448.

[35] S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, and J. Schwenk, "Universally composable security analysis of tls," in *International Conference on Provable Security*. Springer, 2008, pp. 313–327.

[36] R. Canetti, D. Shahaf, and M. Vald, "Universally composable authentication and key-exchange with global PKI," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 9615. Springer, 2016, pp. 265–296.

[37] OpenStack Foundation. (2017, May) Openstack API documentation. [Online]. Available: https://developer.openstack.org/api-guide/quick-start/

[38] M. Jones and D. Hardt, *RFC 6750: The OAuth 2.0 authorization framework: Bearer token usage*. Internet Engineering Task Force (IETF), 2012, https://tools.ietf.org/html/rfc6750.

[39] OpenStack Foundation, "Tokens," 2017. [Online]. Available: https://docs.openstack.org/security-guide/identity/tokens.html

[40] P. Leach, M. Mealling, and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," RFC 4122 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–32, Jul. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4122.txt

[41] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, 2007, pp. 61–85.

[42] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proceedings of the USENIX Winter Conference*. USENIX Association, 1988, pp. 191–202.

[43] H. Krawczyk and H. Wee, "The OPTLS protocol and TLS 1.3," in *IEEE European Symposium on Security and Privacy, EuroS&P*. IEEE, 2016, pp. 81–96. [Online]. Available: https://doi.org/10.1109/EuroSP.2016.18

[44] K. Rarick and T. Maher, 2014. [Online]. Available: https://github.com/fernet/spec/blob/master/Spec.md

[45] OpenStack Identity Team, "Service token composite authorization," March 2016, https://specs.openstack.org/openstack/keystone-specs/specs/keystonemiddleware/implemented/service-tokens.html.

# A Notations Description

Table 1 provides a brief description of notions used in algorithms 8, 26, 29, and 32. Figures 4 and 5 represent the message flow between the ideal cloud $IC$, simulator $S$, and environment $E$ represented in these algorithms. These information aim in guiding readers to understand the algorithms easier.

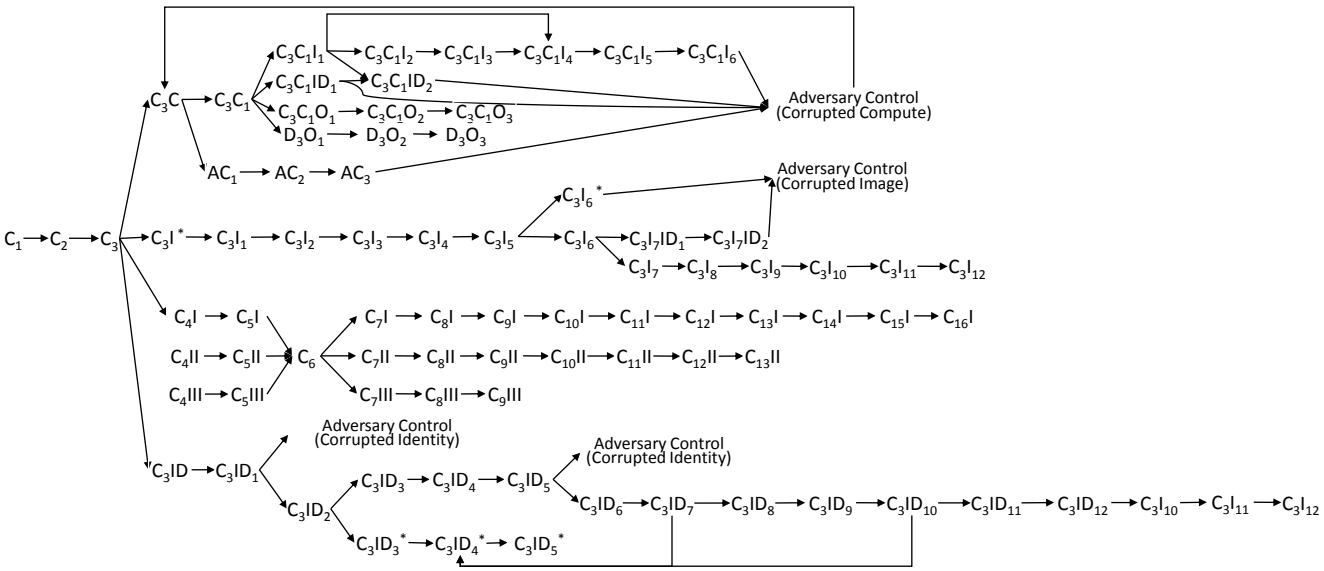


Figure 4: Create Node flowchart which shows the interaction between the ideal cloud, simulator, and environment $E$. This flowchart is valid for both one-time token and bearer token mechanisms. Table 1 describes the notation used in this flowchart.

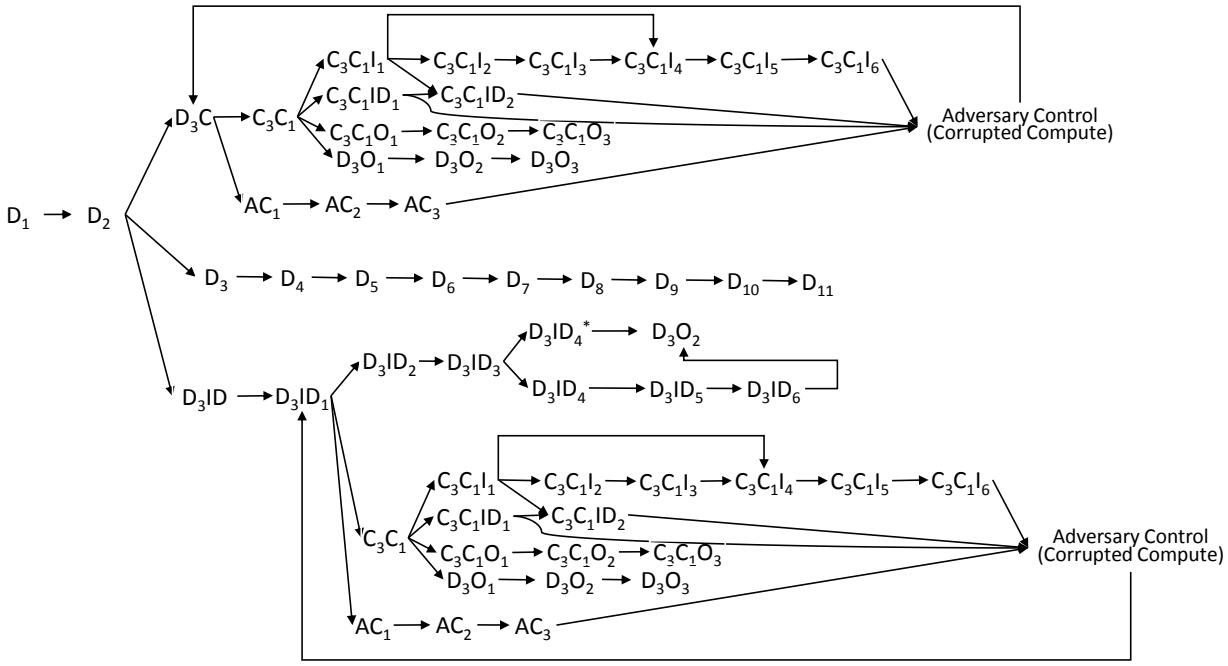


Figure 5: Delete Node flowchart which shows the interaction between the ideal cloud, simulator, and environment $E$. This flowchart is valid for both one-time token and bearer token mechanisms. Table 1 describes the notation used in this flowchart.

Table 1: notation description

| | |
|---|---|
| $C_1$ | Create node request from user. $(IC)$ |
| $C_2$ | $S$ sends the create node request to $E$ and waits for the confirmation. $(S)$ |
| $C_3$ | Request: Create node, Situation: no corruption<br>By receiving confirmation message from $E$, $S$ sends a confirmation message to $IC$. $(S)$ |
| $C_3C$ | Request: Create node, Situation: corrupted Compute<br>$S$ creates a unique credential for the user (if it has not been created before) and together with the message sends it to $\mathcal{A}$ $\mathcal{A}$ and gives the control to $\mathcal{A}$. $(S)$ |
| $C_3C_1$ | Request: Anything, Situation: corrupted Compute<br>$\mathcal{A}$ has made a forward request. Thus, $S$ sends it to $E$ and waits for the confirmation. $(S)$ |
| $C_3C_1I_1$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>$\mathcal{A}$ has made a forward request. The request is to get the corresponding image to image-id. When Image is not corrupted, $S$ sends the request to the ideal cloud to get the result, if the user is allowed to make such request (in one-time token mechanism) otherwise, sends the validation request to $E$. $(S)$ |
| $C_3C_1I_2$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>If Compute is corrupted while Image service is not corrupted and $\mathcal{A}$ makes an Image request. Ideal cloud sends back the result $S$. $(IC)$ |
| $C_3C_1I_3$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>$S$ receives corresponding image to image-id from the ideal cloud. It sends the validation request and waits for confirmation. $(S)$ |
| $C_3C_1I_4$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>After receiving confirmation, the result of the validation is provided to $E$ and waits for confirmation. $(S)$ |
| $C_3C_1I_5$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>After receiving confirmation from $E$, $S$ sends the corresponding-Image of image-id to $E$. And finally after $E$ confirms, the corresponding-Image result will be given to $\mathcal{A}$ (for the corrupted Compute). $(S)$ |
| $C_3C_1I_6$ | Request: Create node, Situation: corrupted Compute, uncorrupted Image<br>After receiving confirmation from $E$, $S$ sends the corresponding-Image of image-id to $\mathcal{A}$. $(S)$ |
| $C_3C_1ID_1$ | Request: validation, Situation: corrupted Compute<br>$\mathcal{A}$ has made a forward request. The request is to validate a request. If Identity is not corrupted, $S$ sees in the DB if this request is compromised and give correct answer, while if the Identity is also corrupted, sends the message to $\mathcal{A}$. $(S)$ |
| $C_3C_1ID_2$ | Request: validation, Situation: corrupted Compute, uncorrupted Identity<br>$S$ sends the validation result to $\mathcal{A}$. $(S)$ |
| $C_3C_1O_1$ | Request: Output create node result, Situation: corrupted Compute<br>$\mathcal{A}$ has made a forward request to show its output to the user. Thus $S$ sends the output result to $E$ and wait for confirmation. $(S)$ |
| $C_3C_1O_2$ | Request: Output create node result, Situation: corrupted Compute<br>After receiving confirmation from $E$, $S$ sends the output result to the ideal cloud. $(S)$ |
| $C_3C_1O_3$ | Request: Output create node result, Situation: corrupted Compute<br>$IC$ outputs whatever $\mathcal{A}$ wants to the user. $(IC)$ |
| $C_3I^*$ | Request: create node, Situation: corrupted Image<br>$S$ sends a message to $IC$ to get more information. $(S)$ |
| $C_3I_1$ | Request: create node, Situation: corrupted Image<br>The ideal cloud needs to get the corresponding-Image from $\mathcal{A}$ if the user is able to create a node. It sends message to $S$. $(IC)$ |
| $C_3I_2$ | Request: create node, Situation: corrupted Image |

| | |
|---|---|
| | $S$ sends validation request to $E$ and wait for confirmation. ($S$) |
| $C_3I_3$ | Request: create node, Situation: corrupted Image<br>By receiving confirmation for $E$, $S$ creates validation result and send it to $E$ and wait for confirmation. ($S$) |
| $C_3I_4$ | Request: create node, Situation: corrupted Image<br>$S$ create a message on behalf of Compute to the Image and request corresponding-Image for image-id and wait for confirmation. ($S$) |
| $C_3I_5$ | Request: create node, Situation: corrupted Image<br>$S$ sends the message the is received by Image to $\mathcal{A}$. ($S$) |
| $C_3I_6$ | Request: anything, Situation: corrupted Image<br>$\mathcal{A}$ sends a message on behalf of Image. ($S$) |
| $C_3I_6^*$ | Request: anything, Situation: corrupted Image<br>$\mathcal{A}$ makes a Get Image request when Image is corrupted. $S$ sends back the result. ($S$) |
| $C_3I_7$ | Request: create node, Situation: corrupted Image<br>If $\mathcal{A}$ has created a message for compute with the image related to image-id (in the previous step), $S$ sends a request to $E$ to get node-id and wait for confirmation. ($S$) |
| $C_3I_7ID_1$ | Request: anything, Situation: corrupted Image\uncorrupted Identity<br>$S$ looks in the DB if this request and creates the proper message, sends to $E$ and waits for confirmation. ($S$) |
| $C_3I_7ID_2$ | Request: anything, Situation: corrupted Image\corrupted Image<br>$S$ sends the messages to $\mathcal{A}$. ($S$) |
| $C_3I_8$ | Request: create node, Situation: corrupted Image<br>By receiving the node-id, $S$ sends the information to $IC$ so it can create the node. ($S$) |
| $C_3I_9$ | Request: create node, Situation: corrupted Image<br>$IC$ creates the node using the corresponding-image and image-id that $S$ provided (if the node-id has not been used before) and sends the result to $S$. ($IC$) |
| $C_3I_{10}$ | Request: create node, Situation: corrupted Image or corrupted identity<br>By receiving the result of create node functionality, $S$ provides the output to $E$ and waits for the Confirmation. ($S$) |
| $C_3I_{11}$ | Request: create node, Situation: corrupted Image or corrupted identity<br>Sends a continue message to the ideal cloud to output the create node result to the user. ($S$) |
| $C_3I_{12}$ | Request: create node, Situation: corrupted Image or corrupted identity<br>$IC$ outputs the result to the user. ($IC$) |
| $C_3ID$ | Request: create node\delete node\access node, Situation: corrupted Identity<br>$S$ creates the validation message to Identity on behalf of Compute and waits for $E$ confirmation. ($S$) |
| $C_3ID_1$ | Request: create node\delete node\access node, Situation: corrupted Identity<br>$S$ sends he result of validation to $\mathcal{A}$. ($S$) |
| $C_3ID_2$ | Request: create node, Situation: corrupted Identity<br>For situation which $\mathcal{A}$ decides to respond to a validation request made from Compute for create node. $S$ sends the validation result to $E$. ($S$) |
| $C_3ID_3$ | Request: create node, Situation: corrupted Identity<br>After $E$ confirms the validation result message if result is successful, $S$ creates a message for Image, to get the corresponding image for image-id. ($S$) |
| $C_3ID_3^*$ | Request: create node, Situation: corrupted Identity<br>After $E$ confirms the validation result message if result is fail, $S$ creates output message on behalf of Compute and sends it to $E$. ($S$) |
| $C_3ID_4$ | Request: create node, Situation: corrupted Identity<br>$S$ creates message on behalf of Image to Identity to get validation result and wait for $E$ confirmation. ($S$) |

| | |
|---|---|
| $C_3ID_4^*$ | Request: create node, Situation: corrupted Identity<br>$S$ sends a message to $IC$ to output the fail result. ($S$) |
| $C_3ID_5$ | Request: create node, Situation: corrupted Identity<br>$S$ sends validation message on behalf of Image for create node request. This message is sent to $\mathcal{A}$. ($S$) |
| $C_3ID_5^*$ | Request: create node, Situation: corrupted Identity<br>$IC$ outputs the create node result that was provided by $S$ to the user. ($IC$) |
| $C_3ID_6$ | Request: create node, Situation: corrupted Identity<br>For situation which $\mathcal{A}$ decides to respond to a request made from Image Service. $S$ sends the validation result to $E$ and waits for confirmation. ($S$) |
| $C_3ID_7$ | Request: create node, Situation: corrupted Identity<br>By receiving the confirmation from $E$ if the Get Image is valid then $S$ sends a message to $IC$ to create the node. If the request is invalid, it creates the output and waits for $E$ confirmation. ($S$) |
| $C_3ID_8$ | Request: create node, Situation: corrupted Identity<br>$IC$ sends the corresponding-Image to $S$. ($IC$) |
| $C_3ID_9$ | Request: create node, Situation: corrupted Identity<br>$S$ receive leaks corresponding image to $E$ and wait for confirmation. ($S$) |
| $C_3ID_{10}$ | Request: create node, Situation: corrupted Identity<br>By receiving confirmation from $E$, $S$ sends a message to get node-id from $\mathcal{A}$. ($S$) |
| $C_3ID_{11}$ | Request: create node, Situation: corrupted Identity<br>By receiving node-id from $\mathcal{A}$, $S$ sends this information to $IC$. ($S$) |
| $C_3ID_{12}$ | Request: create node, Situation: corrupted Identity<br>$IC$ creates the node and provides the result to $S$. ($IC$) |
| $C_4I, C_4II$<br>$C_4III$ | Request: create node, Situation: no corruption<br>$IC$ sends the result of validation and related information for create node to $S$.<br>I: when the user is allowed to create a node and allowed to use image-id.<br>II: when the user is allowed to create a node but not allowed to use image-id.<br>III: when the user is not allowed to create a node. ($IC$) |
| $C_5I, C_5II$,<br>$C_5III$ | Request: create node, Situation: no corruption<br>By receiving information regarding create node validation, $S$ creates a validation request message on behalf of compute to Identity and sends it to $E$. ($S$) |
| $C_6$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, $S$ sends the validation result and waits for confirmation. ($S$) |
| $C_7I, C_7II$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, for I,II: $S$ creates a message on behalf of Compute to Image to get image related to image-id, sends it to $E$ and waits for confirmation.<br>III: $S$ sends the output message to $E$ and waits for confirmation. ($S$) |
| $C_8I, C_8II$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, $S$ creates a validation request on behalf of Image to Identity, sends it to $E$ and waits for confirmation. ($S$) |
| $C_9I, C_9II$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, $S$ creates a validation result on behalf of Identity to Image, sends it to $E$ and waits for confirmation. ($S$) |
| $C_{10}I, C_{10}II$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, for $S$ creates a message on behalf of Image to Compute with corresponding image for image-id, sends it to $E$ and waits for confirmation. ($S$) |
| $C_{11}I$ | Request: create node, Situation: no corruption |

| | |
|---|---|
| | By receiving confirmation message from $E$, $S$ creates a message to request node-id from $\mathcal{A}$ and sends it to $E$. $(S)$ |
| $C_{12}I$ | Request: create node, Situation: no corruption<br>By receiving node-id from $E$, $S$ sends node-id to $IC$. $(S)$ |
| $C_{13}I$ | Request: create node, Situation: no corruption<br>By node-id from $S$, $IC$ creates the node and sends the result to $S$. $(IC)$ |
| $C_{14}I, C_{11}II,$<br>$C_7III$ | Request: create node, Situation: no corruption<br>I: By receiving result message from $IC$, $S$ sends the output message to $E$ and waits for confirmation.<br>II, III: By receiving confirmation message from $E$, $S$ sends the output message to $E$ and waits for confirmation. $(S)$ |
| $C_{15}I, C_{12}II,$<br>$C_8III$ | Request: create node, Situation: no corruption<br>By receiving confirmation message from $E$, $S$ sends confirmation message to $IC$. $(S)$ |
| $C_{16}I, C_{13}II,$<br>$C_9III$ | Request: create node, Situation: no corruption<br>By receiving confirmation message $S$, $IC$ notifies the user about the request result. $(IC)$ |
| $AC_1$ | Request: create node\delete node\access node, Situation: corrupted Compute<br>$\mathcal{A}$ makes a request on behalf of compute. $S$ sends the request to the ideal cloud and waits for the result. $(S)$ |
| $AC_2$ | Request: create node\delete node\access node, Situation: corrupted Compute<br>The $IC$ applies the request if it is possible (without checking validation) and sends the result to $S$ $(IC)$ |
| $AC_3$ | Request: create node\delete node\access node, Situation: corrupted Compute<br>By receiving the result from $IC$, $S$ notifies $\mathcal{A}$ and gives the control to her. $(S)$ |
| $D_1$ | Delete node request from user. The $IC$ sends the request to $S$ $(IC)$ |
| $D_2$ | Request: Delete node, Situation: anything<br>$S$ leaks this message (without credentials) to $E$ and waits for confirmation $(S)$ |
| $D_3$ | Request: Delete node, Situation: no corruption<br>By receiving confirmation from $E$, $S$ sends a confirmation message to $IC$ $(S)$ |
| $D_4$ | Request: Delete node, Situation: no corruption<br>$IC$ validates the user and sends this information to $S$. $(IC)$ |
| $D_5$ | Request: Delete node, Situation: no corruption<br>By receiving information regarding validation from $IC$, $S$ creates a validation request on behalf of Compute to Identity, sends it to $E$ and waits for confirmation. $(S)$ |
| $D_6$ | Request: Delete node, Situation: no corruption<br>By receiving confirmation from $E$, $S$ creates a validation response on behalf of Identity to Compute, sends it to $E$ and waits for confirmation. $(S)$ |
| $D_7$ | Request: Delete node, Situation: no corruption<br>By receiving confirmation from $E$, $S$ sends a confirmation message to $IC$. $(S)$ |
| $D_8$ | Request: Delete node, Situation: no corruption<br>$IC$ checks if node exists and delete it and sends the result to $S$. $(IC)$ |
| $D_9$ | Request: Delete node, Situation: no corruption<br>$S$ creates an output message, sends it to $E$ and waits for confirmation. $(IC)$ |
| $D_{10}$ | Request: Delete node, Situation: no corruption<br>By receiving confirmation from $E$, $S$ sends a confirmation message to $IC$. $(S)$ |
| $D_{11}$ | Request: Delete node, Situation: no corruption<br>$IC$ outputs the result to the user. $(IC)$ |
| $D_3C$ | Request: Delete node, Situation: corrupted Compute<br>$S$ creates a unique credential for the user (if it has not been created before) and together with the message sends it to $\mathcal{A}$ and gives the control to $\mathcal{A}$. $(S)$ |
| $D_3ID$ | Request: Delete node, Situation: corrupted Identity |

| | |
|---|---|
| | $S$ create the "Service Validation" message on behalf of Compute and sends it to $E$ and waits for the Confirmation. ($S$) |
| $D_3ID_1$ | Request: Delete node, Situation: corrupted Identity<br>$S$ sends the message together with the credentials to $\mathcal{A}$. ($S$) |
| $D_3ID_2$ | Request: Delete node, Situation: corrupted Identity<br>If $\mathcal{A}$ decides to respond to a request made from Compute for delete node. $S$ creates a message (with adversarial validation result) on behalf of Identity to Compute, sends it to $E$ and waits for confirmation. ($S$) |
| $D_3ID_3$ | Request: Delete node, Situation: corrupted Identity<br>By receiving the validation result from the corrupted Identity, $S$ generates the fail result if the valid=fail for $E$ and wait for confirmation. While if the valid=success, $S$ sends this information to $IC$ for the execution. ($S$) |
| $D_3ID_4$ | Request: Delete node, Situation: corrupted Identity<br>The $IC$ deletes the node if it is possible and notifies $S$. ($IC$) |
| $D_3ID_5$ | Request: Delete node, Situation: corrupted Identity<br>$S$ receives the result of deleting a node from $IC$, notifies $E$ and waits for confirmation. ($S$) |
| $D_3ID_6$ | Request: Delete node, Situation: corrupted Identity<br>By receiving confirmation message from $E$, $S$ sends a continue message to $IC$. ($S$) |
| $D_3ID_4^*$ | Request: Delete node, Situation: corrupted Identity<br>Sends to ideal cloud the result of the delete node. ($S$) |
| $D_3O_1$ | Request: Output delete node result, Situation: corrupted Compute<br>$\mathcal{A}$ has made a forward request to show its output to the user. Thus $S$ sends the output result to $E$ and wait for confirmation. ($S$) |
| $D_3O_2$ | Request: Output delete node result, Situation: corrupted Compute<br>After receiving confirmation from $E$, $S$ sends the output result to the ideal cloud. ($S$) |
| $D_3O_3$ | Request: Output delete node result, Situation: corrupted Compute<br>$IC$ outputs whatever $\mathcal{A}$ wants to the user. ($IC$) |