Karlsruhe Institute of Technology

Confidential Computing in Distributed Systems

Bachelor's Thesis of

Wenzhe Vincent Cui

at the Department of Informatics
SCC – Steinbuch Centre for Computing

Reviewer:	Prof. A
Second reviewer:	Prof. B
Advisor:	M.Sc. C
Second advisor:	M.Sc. D

11. Month 2021 – 02. Month 2022

I declare that I have developed and written the enclosed thesis completely by myself. I have submitted neither parts of nor the complete thesis as an examination elsewhere. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. This also applies to figures, sketches, images and similar depictions, as well as sources from the internet.

PLACE, DATE

. . . . . .

(Wenzhe Vincent Cui)

Todo list

Abstract

In the current computing landscape distributed computing systems, largely based on Grid and Cloud computing, have become the main ways of sharing infrastructure resources, such as compute, storage, and network resources, while also providing services that ease the development and orchestration of applications on said resources. On the one hand, the usage of these resources and service come with benefits such as higher availability, reducing complexity of applications, and cost-efficiency, on the other hand, traditionally these benefits come with the cost of fully trusting the provider of the resources and services with potentially confidential data. Nevertheless, as consumer and government demands for data privacy increase (e.g., GDPR coming into effect in the EU in 2018), the distributed computing model must adapt to meet the increasingly strict trust requirements. The advent of confidential computing enables a new distributed computing model where the provider of infrastructure resources becomes untrusted by providing hardware-based trusted execution environments (TEEs). This thesis researches the current state of trusted execution environment technologies, remote attestation procedures that allow tenants to verify the trustworthiness of TEEs, and introduces a new *trusted distributed computing model* that integrates these two concepts into the traditional distributed computing model in order to remove the provider of the distributed computing system from the list of trusted parties.

Contents

List of Figures

List of Tables

1 Introduction

Today distributed computing systems are largely based on the Grid or Cloud computing concept. These systems are based on a layered architecture where each layer's security is dependent on the security of the layers beneath it, with the lowest layer being hardware components and the highest layer being applications that implement abstract business logic. Each layer abstracts the complexity of the layers below it and provides a simplified interface to the layers above.

Service providers, entities that provide the resources and services of a distributed computing system, are largely concerned with the protection of the components that make up the lower layers of distributed computing systems (e.g. hypervisors providing virtualization of compute resources) from untrusted code (e.g. virtual machines run by the hypervisor). Traditionally, this isolation only protects the lower layer from the upper layer, and does not protect the confidentiality of the upper layer from the lower layer. As a result, the traditional distributed computing model is completely built around the assumption that tenants trust the service provider's software stack, including privileged software such as hypervisors and firmware, but also the service provider's staff, including system administrators but also those with physical access to hardware. But this trust is not only present in compute, storage, and network resources, today many applications utilize services provided by the service provider that implement privileged functionality, such as authentication and authorization mechanisms and management of cryptographic key material, in order to decrease the complexity of the application and be more cost-efficient.

Currently, the best practice for tenants to protect confidential data is to manage cryptographic keys themselves, for example by operating an own key management service (KMS) or service provider offered hardware security module (HSM). While this solution protects cryptographic keys and therefore encrypted data from the service provider, these keys are still used to decrypt confidential data that is subsequently used by compute resources managed by the service provider, making the protection of cryptographic keys from the service provider pointless. Other solutions rely on new cryptographic primitives that allow specific compute operations to be directly executed on encrypted data [30, 7]. However, these solutions currently are either not applicable to general-purpose computing, or have very high performance limitations that make them impractical.

This thesis introduces the integration of confidential computing technologies into the traditional distributed computing model. Confidential computing is an arising technology that provides execution environments, so-called trusted execution environments (TEEs), that protect applications by utilizing hardware primitives. By implementing the protection of applications in the lowest layer, the hardware, it is possible to remove management software such as hypervisors and operating systems from the list of required trusted software components and service providers from the list of required trusted entities. While TEEs provide protection for applications and confidential data, tenants have to be able to verify that applications are actually running inside TEEs and that the platform that provides TEEs has not been modified by the service provider. Remote attestation is an already widely used method for assessing the trustworthiness of remote devices, components, and environments. Commercially available confidential computing technologies such as Intel SGX and AMD SEV have built-in support for producing evidence that can be used for remote attestation.

This thesis will first research the current state of the traditional distributed computing model, confidential computing technology, and remote attestation procedures in Chapter 3 and subsequently discuss the new *trusted distributed computing model* in which service providers are considered untrusted in Chapter 4. Section 3.1 decomposes the two major distributed computing models Cloud and Grid computing into a generalized architecture, which is evaluated by Section 4.1 in order to identify threats from untrusted service providers defining a threat model. Chapter 4 is mainly concerned with the integration of confidential computing (Section 3.2) and remote attestation (Section 3.3) into the traditional distributed computing model and defines requirements of the resulting trusted distributed computing model in Section 4.4. Lastly, this thesis will evaluate the untrusted distributed computing model upon the threat model (Section 4.5), and look at two case studies (Section 4.6).

2 Background

2.1 Cryptographic Concepts

Cryptography is the practice of protecting data by transforming it into a format that is unreadable by an unauthorized party. In this background section we will have a look at the basics of symmetric cryptography, asymmetric cryptography, and key agreement protocols.

2.1.1 Symmetric and Asymmetric Cryptography

Symmetric cryptography, also known as secret-key cryptography, is a method of encryption where the same key is used for both encryption and decryption. This means that both the sender and receiver must have the same key to encrypt and decrypt messages. In symmetric cryptography, the key is kept secret and secure to prevent unauthorized access. The most common symmetric encryption algorithms are Advanced Encryption Standard (AES) and Data Encryption Standard (DES).

The main benefit of symmetric cryptography is its performance. Since the same key is used for encryption and decryption, it is a fast and efficient process. However, one of the major drawbacks of symmetric cryptography is the issue of key management. Managing and distributing keys to entities and components that require access to encrypted data is a challenging task. Furthermore, if a key is compromised, all data encrypted with that key can be easily decrypted.

Asymmetric cryptography, also known as public-key cryptography, is a method of encryption where two different keys are used for encryption and decryption. These keys are mathematically related, but they cannot be derived from each other. One key is kept private, and the other key is made public. The private key is used for decryption, while the public key is used for encryption. The most common asymmetric encryption algorithms are RSA and Elliptic Curve Cryptography (ECC).

One of the significant benefits of asymmetric cryptography is that it eliminates the need for key management. Each user or system can have a unique pair of public and private keys, and the public key can be shared with anyone. The private key, on the other hand, is kept secure and confidential. Asymmetric cryptography is slower

than symmetric cryptography due to the complexity of the mathematical algorithms involved. However, it is more secure since even if the public key is compromised, the private key cannot be derived from it.

Digital signature is a widely used application of asymmetric cryptography. A digital signature is a cryptographic primitive that is used to verify the integrity and authenticity of a data. The process of digital signature involves two steps: signing and verification. To sign data, the owner of the data uses their private key to generate a unique digital signature. When the owner sends the data together with the signature to another party, the receiver can use the owner's public key to verify the signature's authenticity and the data's integrity. If the verification of the digital was successful, the receiver can be confident that the data was sent by the owner has not been modified during transit.

2.1.2 Key Agreement

Key agreement is a crucial component of secure communication. It involves two parties agreeing on a secret key (symmetric cryptography) that can be used for encryption and decryption of messages. One of the most popular key agreement protocols is the Diffie-Hellman protocol.

The Diffie-Hellman protocol is a key agreement protocol that utilizes asymmetric cryptography in order to allow two parties to agree on a mutual secret key over an insecure channel. A simplified overview of the process is as follows (with parties Alice and Bob):

1. Both parties establish shared parameters: a prime number p and generator g.
2. Each party generates a random number (A for Alice and B for Bob) and computes a publicly known number $pub_A = g^A \mod p$ and $pub_B = g^B \mod p$ respectively.
3. Both parties exchange their public number pub_A and pub_B.
4. Both parties compute the shared secret key $K = (pub_A)^B = (pub_B)^A = g^{AB} \mod p$
5. The secret key K can then be used to encrypt and decrypt messages send between both parties, establishing a secure communication channel over an insecure channel.

Even though attackers might gain knowledge about p, g, pub_A, and pub_B by listening to the insecure channel, they can not derive the secret key K from the publicly known numbers. If we assume that the prime number p and generator g are already specified in a specific protocol or have been pre-established, there are exactly two messages that need to be exchanged between Alice and Bob. For example, if Alice is the initiator of

the communication, Alice starts by sending pub_A upon which Bob responds with pub_B. After exchanging those two messages, Alice and Bob can already derive the shared secret key K.

However, an attacker might perform a man-in-the-middle attack, where the attacker intercepts the key agreement process and establishes shared secrets with both Alice and Bob. If Alice sends Bob an encrypted message, the attacker decrypts the message using the shared secret key with Alice, gains access to the plain text content of the message, encrypts the message using the shared secret with Bob, and finally relays the message to Bob.

Furthermore, the attacker can even impersonate either party in the communication. For example, Bob thinks that the secret key he/she established with the attacker is only known by Bob and Alice. Therefore, Bob assumes that messages encrypted with this secret key and send to Bob by the attacker, are sent by Alice.

In order to mitigate man-in-the-middle attacks, either Alice or Bob has to authenticate the other party. This can be achieved using asymmetric cryptographic keys. For example, Bob uses his own private key to sign the message that includes pub_B and is sent to Alice. Alice then uses Bob's public key in order to verify that the message is indeed sent by Bob. The public keys can be securely exchanged between Alice and Bob by facilitating a Public Key Infrastructure (PKI). However, this thesis will not further go into detail about PKIs.

3 Technical Research

3.1 Traditional Distributed Computing Model

Distributed computing systems are inherently distributed systems, which means they share the fundamental characteristics and goals, defined by Steen and S. Tannenbaum [39] as resource sharing, transparent distribution, openness, and scalability. Distributed computing facilitate distributed system principles in order to group a set of resources (possibly at a geographical distance) and provide tenants a single, coherent view of these resources. In general distributed computing systems allow users to share, manage the access to, and use compute, storage, and network resources.

In the following two sections we will look at two major distributed computing concepts, Grid and Cloud computing.

3.1.1 Grid Computing

The Grid Computing model focuses on sharing widely distributed compute resources in order provide problem-solving environments and enable collaboration for a set of individuals or institutions, so-called virtual organizations. It provides basic mechanisms, in the form of network protocols and interfaces. These protocols and interfaces offer means of discovering, managing the access to, and using remote resources. Because those resources are not subject to centralized control, these protocols and interfaces have to be standardized and open to enable interoperability [16].

The hourglass model facilitated by the internet protocol stack can also be found in the Grid computing architecture. While the base of the hourglass provides various fundamental behaviors, the neck defines a small set of abstractions, allowing a diverse set of high-level behaviors to be built on top. The four layers of the Grid computing model are as follows:

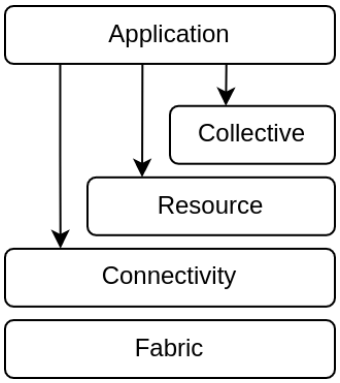

Figure 3.1: The layered Grid architecture.
Source: Foster, Kesselman, and Tuecke [16]

Fabric Layer

The Fabric layer provides the resources which are shared by Grid protocols and offers local, resource-specific operations that occur on specific resources as a result of sharing operations at higher levels. This should at least provide resource enquiry mechanisms, enabling the discovery of their structure, state and capabilities, and resource management mechanisms, providing control over them.

Connectivity

This layer specifies core communication and authentication protocols, enabling the exchange of data between Fabric layer resources and providing secure mechanisms for verifying the identity of users and resources. Communication usually requires transport, routing, and naming protocols.

Resource

On top of the Connectivity layer the Resource layer defines protocols providing secure negotiation, initiation, monitoring, control, accounting, and payment mechanisms for individual resources. It utilizes Fabric layer functions in order to access and control local resources. These protocols are focused on a single resource, implying that they ignore issues of global state and atomic actions across distributed collections.

Collective

While the Resource layer is concerned with the interactions of a single resource, the Collective layer focuses on the interaction of a collection of resources. Being the top and building on the neck (Connectivity and Resource layer) of the hourglass allows this layer to provide a wide variety of behaviors, such as discovering, co-allocation, scheduling, brokering, monitoring, and replication of resources.

Applications
The final layer of the Grid architecture is the collection of user applications. These applications implement specific business logics by utilizing resources and services of the previous layers.

3.1.2 Cloud Computing

Cloud Computing systems usually have a centralized control and facilitate both open and proprietary protocols and interfaces in order to provide on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured services. *The NIST Definition of Cloud Computing* [33] defines three distinct service models of cloud computing:

Infrastructure as a Service (IaaS)
In an IaaS service model the managed resources are fundamental computing resources. This includes – physical or virtual – machines, storage, and networks. The consumer does not manage or control the underlying infrastructure but can deploy and run arbitrary software, including operating systems, onto the provided infrastructure.

Platform as a Service (PaaS)
The PaaS service model adds another layer of abstraction on top of the IaaS service model. Consumers can deploy and run applications on a provided platform that supports a set of programming languages, libraries, services, and tools. So the managed resources in a PaaS service model are applications. The consumer does not manage or control the underlying infrastructure, and additionally has no control over the operating system.

Software as a Service (SaaS)
In this model the managed resources are applications. But in comparison to the PaaS model the consumer is not able to run arbitrary applications, but only a provider specific selection of applications. The customer does not manage or control the underlying infrastructure, operating system, or even the application.

These service models provide tenants with resources that applications run on and services that support the development and deployment of these applications.

3.1.3 Trust Model

There are three roles that are present in the traditional distributed computing model.

Service Provider

In general a service provider is an entity or organizational unit that provides services to other entities or organizational units. In the distributed computing context service providers provide application owners and data owners with infrastructure resources and/or platform services.

Application Owner

Application owners manage applications that operate on data owned by the data owner. This does not imply that the application owner develops applications. Applications can be developed and provided by separate entities.

Data Owner

Data owners are in the possession of data that used, and/or manipulated by an application. They are concerned about the confidentiality of their data. If there are multiple data owners there may be mutual distrust between the data owners.

This trust model needs to be applied from the perspective of the data owners and is tied to a set of data. For example, in a Grid environment, there are three virtual organizations A, B, and C, all sharing resources and services. Assuming that virtual organization A owns a dataset D, but all of A's resources are currently occupied. Therefore, A wants to use resources from virtual organization B in order to run an application on dataset D. In this context, while all virtual organizations share resources, only B takes on the role of a service provider, because A does not share resources used for this specific context and C is not involved at all. Because A manages the application and owns the dataset D, A takes on the role of application owner and data owner in this example.

Traditionally, the data owner trusts both the application owner and the service provider. Therefore, both roles can be taken on by the same party.

3.1.4 Architectural Overview

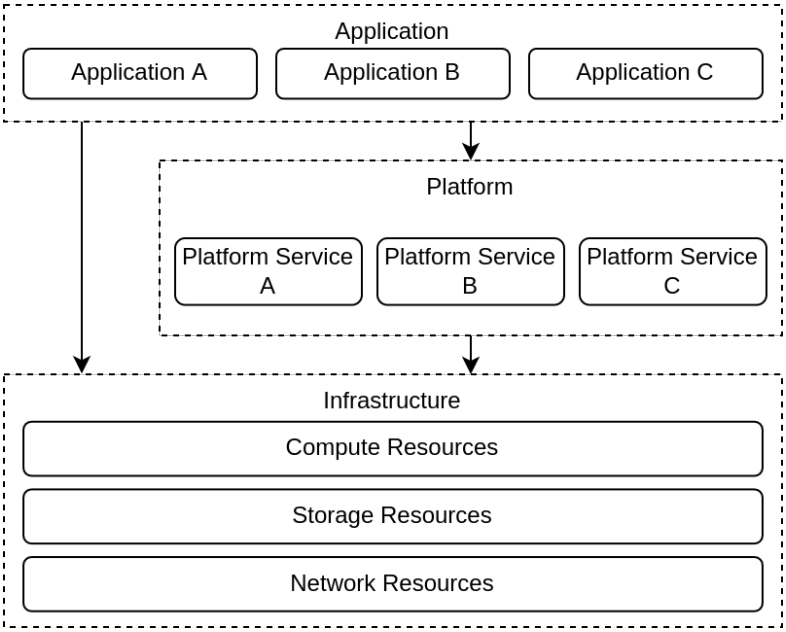

Figure 3.2: Layered model of a distributed computing system.

This section defines a generalized architecture of the distributed computing model, which consists of three layers.

Infrastructure

This layer provides fundamental resources, such as compute, network, and storage resources. This commonly includes physical or virtual machines (compute resource), layer two or three networks connecting machines (network resource), and network attached storage (storage resource).

In the Cloud model this corresponds to the IaaS service model and in the Grid architecture it is represented by the Fabric layer.

Platform

The Platform layer sits between the Infrastructure and Application layer. It provides a set of services and tools that support application owners to run applications in a distributed computing environment and manage the underlying infrastructure resources required for the application to run. The platform layer abstract the complexity of the underlying infrastructure by providing services to manage,

access, and utilize needed resources. These services are all operated and provided by the service provider.

This layer matches the Cloud's PaaS and SaaS service model and is represented by the collection of the Resource, Connection, and Collective layer in the Grid architecture.

Application

The final layer is the collection of applications and services managed by application owners.

In the following figures components of the Infrastructure layer are marked in blue, Platform layer in red, and Application layer in green.

Infrastructure Layer

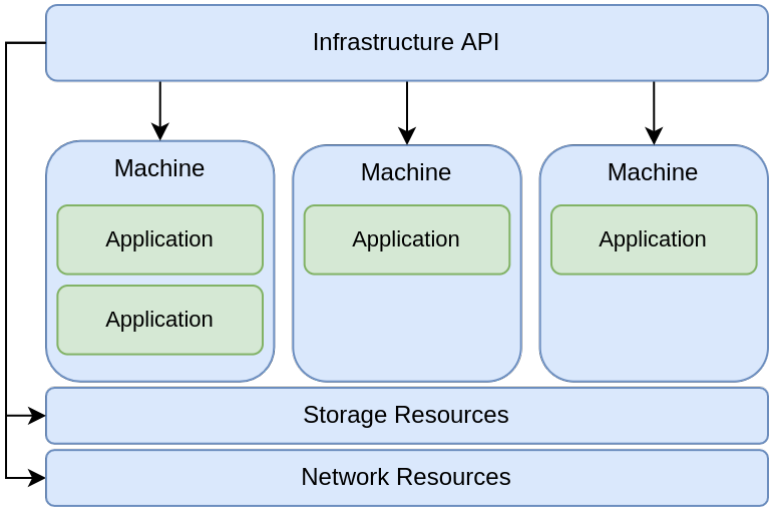

Figure 3.3: Overview of the Infrastructure layer in the traditional distributed computing model.

Typically, computing resources provided by the Infrastructure layer are in the form of physical or virtual machines. These machines combine CPUs, memory, and peripheral devices. On the other hand, storage and network resources can be provided in various forms, such as block-level, file-level, and object-level storage, and layer two and three networks. In order to keep this model abstract, we will not make any assumptions about specifics of the storage and network resources. This model will assume the presence of an application programming interface (API), that can be used to implement a user

interface. While various kinds of user interfaces can be provided to tenants, such as websites, web portals, terminal user interfaces (TUIs), graphical user interfaces (GUIs), software development kits (SDKs), and libraries, these user interfaces typically facilitate an API that exposes resource management actions.

As outlined before, there are two types of resources in the Infrastructure layer: physical and virtual. Physical Infrastructure layer resources protection is implemented by the service providers. For example by securing the physical location of the hardware and utilizing application transparent encryption. Virtual infrastructure is protected by the isolation that virtualization systems provide. This requires a correctly configured and actively patched virtualization system, as misconfiguration can lead to breaking the isolation guarantees, and not patching virtualization systems allows attackers to exploit vulnerabilities in those implementations. For example hypervisors are large pieces of software that require complex configuration, and over the years numerous vulnerabilities have been found in commonly used hypervisors [32, 34]. These virtualization systems are generally managed and controlled by the service provider. Therefore, protection of infrastructure layer resources, regardless of whether the resources are physical or virtual, is entrusted to the service provider.

Platform Layer

The platform layer offers various kinds of services. Commonly found services are:

Resource management
: Management of compute and storage resources. This could include the ability to (co-)allocate and release resources in order to dynamically scale depending on the current needs, and monitor resource usage.

Authentication and authorization
: Providing ways to verify the identity of a user or process and define and enforce policies governing the access to infrastructure resources and applications.

Messaging and communication
: While the infrastructure provides basic infrastructure for communication by providing network resources, the platform layer often provides higher level communication such as inter-process communication, message passing, and event notifications.

Data management
: Providing means for storing and accessing data by managing storage resources of the Infrastructure layer. Often also provides caching, replication, synchronization services in order to maximize availability, integrity, and performance.

Service discovery and load balancing
Exposing applications as network services to other applications and distributing traffic to multiple instance of an application.

Monitoring and logging
Support the monitoring of applications for failures and performance issues and aggregate logs of applications for debugging and auditing purposes.

Collaboration frameworks
Provide problem-solving environments that manage multistep, asynchronous, multi-component workflows.

Application deployment
Decrease the burden of deploying applications, providing mechanisms to deploy and configure applications in execution environments.

Key Management
A Key Management Service (KMS) manage cryptographic keys used to encrypt and decrypt confidential data. These services typically provide mechanisms for generating, storing, and providing keys to other applications and services.

While these services implement various types of behaviors, they can be grouped into five non-mutually exclusive categories:

Infrastructure management
Services that manage Infrastructure layer resources in order to simplify the management and access to those resources, and provide more complex behaviors, such as data management, application-level communication, service discovery, and load balancing. These services need the privilege to (co-)allocate, release, and monitor compute, storage, and network resources.

Security
Services that offer application-level authentication, authorization, and/or encryption, easing the process of implement security into an application. These services manage identities of applications and users, control the access to resources and services, and generate and manage keys, used for encrypting and decrypting data.

Monitoring and Logging
Monitoring and aggregating logs of applications in the Platform layer is typically handled by the service provider. While monitoring metrics usually do not contain sensitive information, the application logs can in some cases contain sensitive information.

Application orchestration

Application orchestration refers to the automated management of applications. This involves the preparation of a target execution environments, and installing, configuring, and executing application inside the target environments.

The usage of Platform layer services provided by a service provider is optional, an application owner can manually manage Infrastructure layer resources, collect logs and metrics, and orchestrate applications or implement those services in the Application layer. But because the service provider is assumed to be trusted, it is often in the interest of the data and application owner to facilitate services provided by the service provider, in order to be more cost-efficient and reduce the complexity of applications.

Application Layer

The final layer is the Application layer, which consists of applications that are managed by the application owner. These applications can also be in the form of services that are typically located in the Platform layer. For example the application owner can choose to manage its own security service that provides authentication, authorization, and encryption to other applications.

The main distinction between the Application layer and the Platform layer is that the applications of the Application layer are managed by the application owner, while the Platform layer services are operated by the service provider.

3.1.5 Example: Kubernetes in a Cloud Environment

Kubernetes is an extensible, open source platform for managing containerized applications. Due to its open source nature and popularity there has been a rapidly growing ecosystem surrounding Kubernetes. It provides general platform services that fall in the infrastructure management and application orchestration, with the ability to integrate with monitoring and logging solutions. While Kubernetes provides the services of the Platform layer, it manages applications on top of Infrastructure layer resources.

Commonly used cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer a managed Kubernetes service (AWS Elastic Kubernetes Service[1], Azure Managed Kubernetes Service[2], Google Kubernetes Engine[3]), providing both Infrastructure resources and Platform layer services. In this example the cloud provider takes on the roles of the service provider.

[1] `https://docs.aws.amazon.com/eks/index.html`

[2] `https://learn.microsoft.com/en-us/azure/aks/`

[3] `https://cloud.google.com/kubernetes-engine/docs`

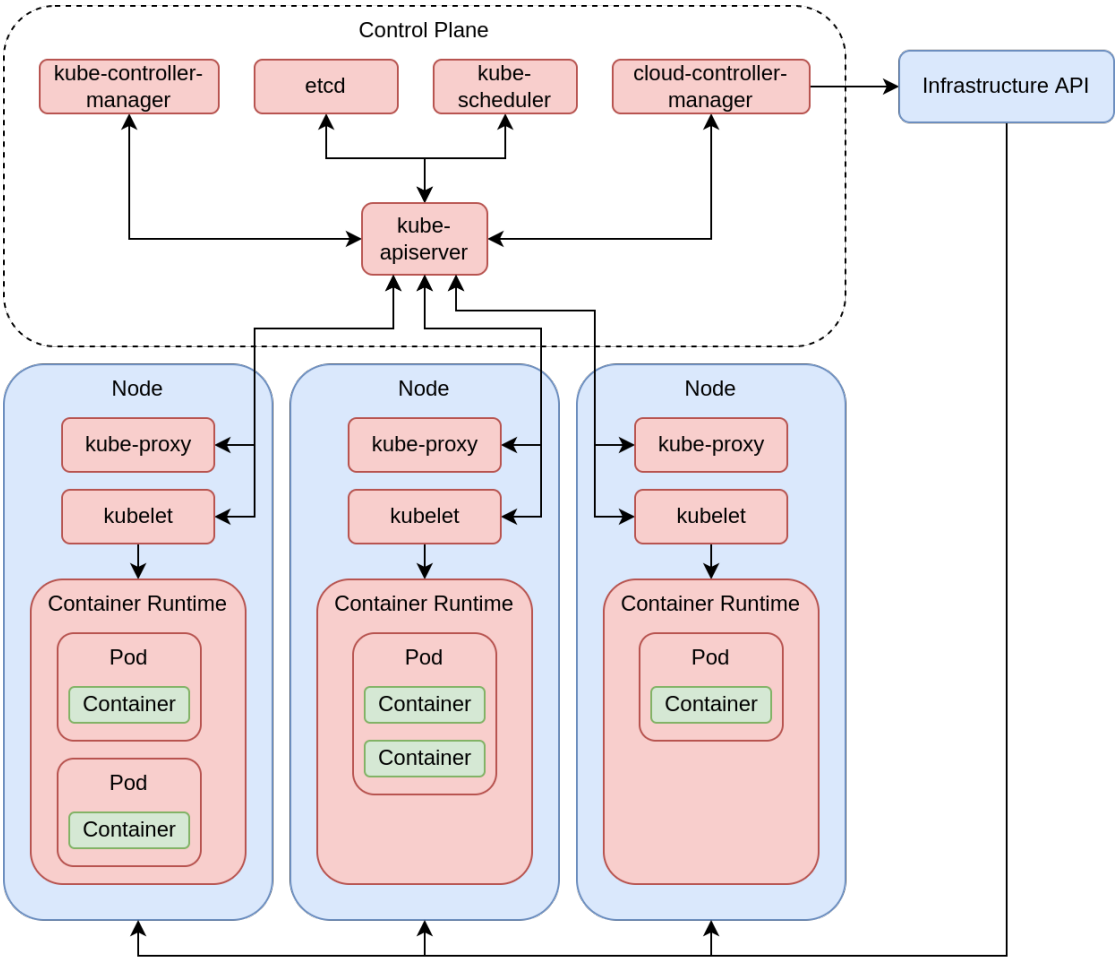

Figure 3.4: Overview of core Kubernetes components.

We will first have look at an overview over the components of a Kubernetes cluster. Figure 3.4 can assist the reader in understanding the relations between those components.

Pods A group of containers that share storage and network resources that models an application-specific logical host. The containers that make up a pod are always located on the same node and are scheduled in unison.

In a broader sense, the pods are execution environments in which applications, in the form of containers, can be deployed and executed. As such, the pods are still part of the Platform layer, execution environments managed by an application

orchestration service, and the containers are the applications of the Application layer.

Nodes

Nodes are machines running containerized applications. On each node run the following components: kubelet, container runtime, kube-proxy. The kubelet is an agent that is responsible for making sure that all containers of a pod are running, while the container runtime is the software that is actually responsible for running containers. The kube-proxy component allow the discovery of applications running inside the cluster and the exposure of those applications as services.

These nodes correspond to compute resources of the Infrastructure layer. The components on those nods however are part of the Platform layer, as they implement specific functionalities needed in order to provide the services of the Platform layer.

Control Plane

A collection of components that manage nodes and pods inside the cluster, making global decisions about the cluster. This includes an API (kube-apiserver), a backing store (etcd) for all cluster data, and a scheduler (kube-scheduler) that assigns pods to nodes. Kubernetes adopts the concept of control loops, enabling the use of so-called controllers, which are non-terminating loops that watch the state of the cluster and make requests of changes when needed. While each of these controllers are logically separate processes, they are compiled into a single kube-controller-manager component. The management of Infrastructure layer resources is abstracted by the cloud-controller-manager which implements cloud specific resource management. All control plane components are also deployed in the Kubernetes cluster using pods.

The components of the control plane implement specific behaviors, which together with the components on the nodes, provide the services of the Platform layer.

Infrastructure Management and Security

Kubernetes enables cloud providers to implement cloud specific infrastructure management services by developing a cloud-controller-manager, Container Storage Interface (CSI), and Container Network Interface (CNI), allowing tenants to automatically scale their Kubernetes cluster depending on the current utilization of infrastructure resources in order to be more cost-efficient.

The cloud-controller-manager typically consists of multiple components that create and update load balancers, and manage the lifecycle of nodes.

CSI plugins are responsible for creating, updating, and destroying storage resources in order to provide pods with persistent storage, referred to as persistent volumes. Cloud providers often also provide storage encryption services in conjunction with their storage resources. These encryption services use keys provided by a KMS. While some cloud providers allow tenants to provide their own KMS, the encryption service still needs plain text access to keys in order to encrypt and decrypt data.

CNI plugins create, update, and destroy the underlying network resources of the Kubernetes cluster and provide node-level, and pod-level communication. Some CNIs also implement authentication and authorization that allow tenants to enforce network policies in the cluster, and transparent encryption, protecting confidential data traversing network resources.

Application Orchestration

The basic workflow of how Kubernetes deploys, configures, and starts a pod on a node is as follows (see Figure 3.5). Note that this description of the workflow is greatly simplified.

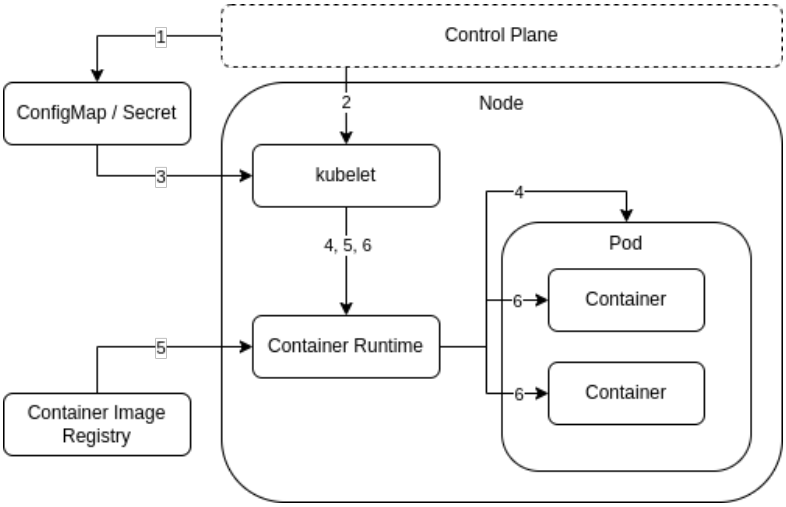

Figure 3.5: Kubernetes' application deployment, configuration, and execution workflow.

1. Upon request of a tenant, the control plane creates ConfigMaps/Sects, which contain non-confidential/confidential configurations of an application.
2. Upon request of a tenant, the control plane requests the kubelet of a node to create a pod and provides the kubelet with the specification of the pod. This spec-

ification includes container images, ConfigMaps/Secrets, storage configuration, and network configuration.

3. Kubelet pulls specified ConfigMaps/Secrets onto the node.
4. Kubelet calls the container runtime to create the pod, configure its storage and network, and provide the pod with the configurations of the
5. Kubelet calls the container runtime to pull the specified container images.
6. Kubelet calls the container runtime to create and start the application containers inside the pod using the pulled container images.

3.2 Confidential Computing (CC)

Data can be in three distinct states: “at rest”, “in transit”, and “in use”. These three states describe data that are stored in persistent storage, traversing a network, and data that is currently being processed. While technologies protecting data “at rest” and “in transit” are commonly used today, there are not many methods to protect data “in use”.

By executing computations in hardware-based trusted execution environments (TEEs) confidential computing protects data in use. In order to for a service provider to be able to assure an application owner that the requested environment can be trusted, remote attestation protocols (see Section 3.3) are used.

The goal of confidential computing is to reduce the size of the trusted computing base (TCB) of applications. The TCB of an application or system is the set of all hardware, firmware, and software components that are critical to its security.

3.2.1 Trusted Execution Environments (TEEs)

Properties

There are different definitions of a trusted execution environment (TEE) with varying properties. The three main properties defined by the Consortium [12] are:

Data confidentiality
: Prevent unauthorized entities to view data that is in use within a TEE.

Data integrity
: Prevent unauthorized entities to add, remove, or change data while it is in use within a TEE.

Code integrity

Prevent unauthorized entities to add, remove, or change code executing in the TEE.

Combined, these three properties not only ensure the confidentiality of the data but also allow clients to trust the results of a computation running inside a TEE using the provided data.

TEEs often also provide evidence in the form of measurements of its initial and/or current state. This evidence can then be verified by a remote party and can help to decide whether to trust the TEE to provide the three properties defined above. Typically, this evidence is cryptographically signed by hardware, allowing the third party to verify the authenticity and integrity of the evidence. This process is referred to as remote attestation, and we will look the specifics of this process in Section 3.3.

Hardware Support

The security of a software layer can only be as strong as the layers below it. This is why an ideal security solution acts from the lowest layer possible. By providing security through the lowest layer – the hardware – it is possible to remove all software layers between the hardware and the TEE from its TCB, including system software such as the operating system or hypervisor. The only component remaining in the TCB is the hardware providing the TEE properties.

Today most TEE implementations still rely on firmware components. This allows manufacturers to more easily deploy bug fixes and security patches. While firmware is still software, these TEE technologies provide hardware based mechanisms that allow third parties to not only verify the trustworthiness of the TEE, but also verify the configuration and integrity of the firmware and software components involved in the creation and management of a TEE.

Memory Protection

Most TEE technologies today rely on the protection of memory in order to provide the three properties defined above. They often provide two mechanisms, protecting the confidentiality and integrity of data stored in memory:

Memory Encryption

TEE technologies rely on a hardware component to encrypt data that is being transferred from the CPU to the physical memory of a machine and decrypt data moving from the memory to the CPU. Unlike homomorphic encryption, which provides specific computational functions directly on encrypted data [30], TEE

technologies transparently en-/decrypt data. Memory encryption strengthens the confidentiality of data in use, as untrusted software components that gain access to the memory of a TEE or malicious entities that have access to the physical memory of a machine, only see encrypted data.

Memory Access Control

On the other hand, memory integrity is guaranteed by enforcing that only the VM or process owning specific memory regions to be able to modify data stored in those. This is achieved by introducing new CPU instructions or a software component that enforces access control.

3.2.2 TEE Models

There are two distinct models of TEEs, process-based and VM-based.

Process-based TEEs

Process-based TEEs introduce a new programming model. A program needs to be split into two components, trusted and untrusted. These are often referred to as the "enclave" and "host". The enclave is executed in a TEE and as such should contain all code that interacts with sensitive data, whereas the host component is responsible for handling non-sensitive tasks like networking and file I/O.

While the host is not shielded the enclave is protected from the rest of the system, this includes

- the enclave's own host
- other processes running on the same machine
- the operating system
- firmware such as the BIOS
- the hypervisor and host operating system (in virtualized environments)
- hardware other than the processor

Splitting a program into enclave and host is challenging. It requires a deep understanding of security and how these process-based TEE solutions work. To ease the development of such applications SDKs and frameworks often hide the split between host and enclave from the developer [38].

Library OSes like Gramine and Occlum go even further and provide a POSIX-like runtime environment with support for network, file I/O, and multithreading. Because

applications running inside the enclave do not have access to the underlying OS, library OSes provide libraries that implement OS system calls in form of library functions. These libraries then interface with a boilerplate host for I/O [41].

Even though these SKDs, frameworks, and library OSes ease the development and make porting of existing applications easier, using process-based TEEs still requires more development effort and more often then not modifications of existing applications.

VM-based TEEs

The main concept of VM-based TEEs is to apply the TEE properties to a full virtual machine. While traditional VMs can share memory with the hypervisor in order to support communication between VMs or with the hypervisor, the memory of VM-based TEEs are not accessible by the hypervisor.

While this model allows running basically any application without modifications, this model has a much larger TCB compared to the process-based TEE model. VM-based TEEs however are still specifically designed to remove the following components from the VMs TCB:

- VM firmware (e.g. OVMF)
- the hypervisor and host operating system
- hardware other than the processor

Comparison

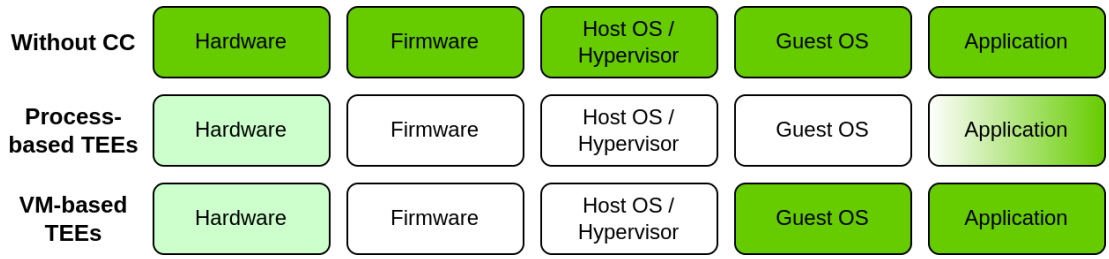

Figure 3.6: Comparison of trusted execution environment models in a virtualized environment.

Figure 3.6 shows a simplified comparison between the TCBs of an application running without confidential computing, inside a process-based TEE, and inside a VM-based TEE. In both TEE models a hardware component enforces the TEE properties and thus

the hardware has to be partly trusted. The main difference between the two models is the size of the TCB, where process-based TEEs only have a small trusted component of the application (the enclave), the whole VM is trusted in a VM-based TEE.

3.2.3 Commercially Available TEE Technologies

Intel SGX

Intel Software Guard Extensions (SGX) provides process-based TEEs by relying on hardware to establish enclaves that contain application code and confidential data. As the name implies, it is an extension of Intel's CPU instruction set architecture [13].

The CPU protects a designated memory area called the Processor Reserved Memory (PRM) established by SGX, by ensuring that other software, such as the system software (hypervisor or OS) and DMA devices do not have access to the PRM. Confidential data and code of enclaves is stored in the Enclave Page Cache (EPC), which is a subset of the PRM. SGX relies on an untrusted system software to manage EPC pages by assigning EPC pages to enclaves and evicting these pages if needed. However, system software cannot directly access the EPC and the CPU maintains Enclave Page Cache Map (EPCM) in the EPC that keeps track of allocated EPC pages and the enclave which owns the page. Using the EPCM the CPU checks the correctness of the system software's allocation decisions, ensures that a EPC page is only assigned to a single enclave, and that only the assigned enclave can access and modify the EPC page. The CPU also encrypts EPC pages while they are stored in physical memory in order to prevent leaking confidential data through PME attacks and guaranteeing confidentiality of EPC pages after eviction.

Initially, the system software ask the CPU to copy data from unprotected memory into EPC pages and assigns the pages to the enclave. After the EPC pages are loaded, the enclave is marked as initialized and the system software can not access nor modify EPC pages anymore. The CPU then measures SGX components and the initial EPC pages of the enclave, producing an attestation report which is then signed by the CPU. The signature can subsequently be used to verify the authenticity of the measurements, and the measurements to verify the integrity of the enclave.

AMD SEV

AMD SEV-SNP is the latest iteration of AMD's Secure Encrypted Virtualization (SEV) technology [3, 4, 5] and as the name implies provides VM-based TEEs.

SEV relies on hardware embedded encryption engines that encrypt or decrypt memory pages written to or read from the physical memory of a machine. It utilizes the AMD Secure Processor (AMD-SP), which is integrated into the same chip as the CPU, to generate and manage cryptographic keys used for the en-/decryption. All software and data is tagged with an Address Space Identifier (ASID). The CPU uses the ASID to restrict the usage of data to the owner with the same ASID and protect the data from any unauthorized usage inside the CPU. However, in the first iteration of SEV the registers of a vCPU could be used to leak confidential data when shutting down a VM. Subsequently, AMD released their second iteration SEV-ES (Encrypted State), which not only encrypted VM memory but also the vCPU's registers. The latest version SEV-SNP (Secure Nested Paging) introduced further features in order to protect the integrity of VM memory.

The attestation process for SEV VMs is similar to the attestation process of SGX enclaves. A hypervisor launches a VM and after the VM is fully loaded the VM's memory is encrypted. After which the AMD-SP measures SEV components and VM memory pages and signs these measurements. Again, the signature can then be used for verifying the authenticity of the measurements, and the measurements to verify the integrity of the VM. Attestation support in SEV and SEV-ES was limited, as measurements could only be requested during the launch of a VM. SEV-SNP supports the request of measurements at any time, enabling more flexible attestation.

As TEE technologies widely differ in their implementations and TEE model they offer, following this section we will treat the hardware and software components that create and protect TEEs as a single platform, which consists of all hardware and software components relevant in the TEE create and protection process, and refer to it as the "TEE platform".

3.2.4 Limitations

Performance Impact

Generally, currently existing solutions either require careful configuration in order to achieve acceptable performance or are inappropriate for specific types of workloads (e.g. high performance computing) [1].

CPU centric focus

Most of today's CC solutions focus on a CPU-level view of memory permissions. This limits the application of CC to heterogeneous computing systems, where heterogeneous accelerators are used in order to speed up specific computations

(e.g. NPUs for machine learning workloads [10]). However, there is ongoing work on integrating CC into heterogeneous computing systems [21].

New technology that requires further research

Since the introduction of Intel SGX in 2015 numerous vulnerabilities found in the SGX architecture [15]. AMD SEV has also not been spared, which until now required two iterations to fix the issues that have been discovered. Both technologies also depend on existing software such as hypervisors, VM firmware, and operating systems to implement the integration of these technologies. These implementations also require further research and testing in order to find vulnerabilities and security issues in their designs.

3.3 Remote Attestation

Remote attestation is the process that allows a remote party to verify the trustworthiness of a device or platform. *Remote ATtestation procedureS (RATS) Architecture* [8] provides a standardized framework for remote attestation, which defines various roles, protocols, and data structures involved in this process. This section will provide an overview of the remote attestation process.

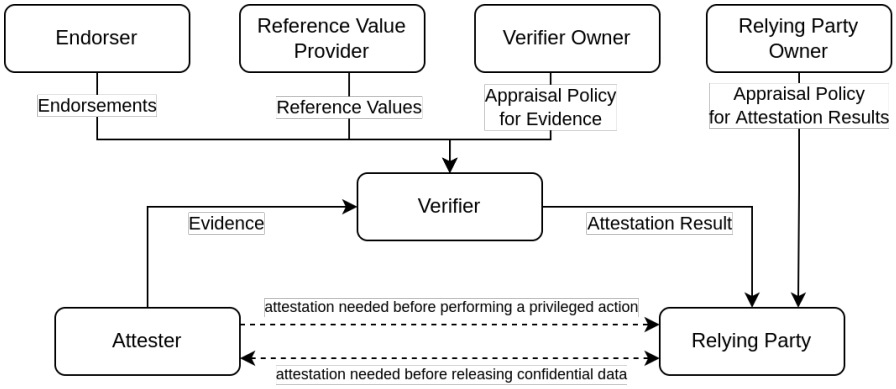

Figure 3.7: Remote attestation roles and data flow.

The remote attestation process begins with the relying party requesting a verification of the attester. Subsequently, the attester generates evidence about its trustworthiness, which is used by the verifier in combination with endorsements from endorsers and reference values from reference value providers by applying appraisal policies to assess the trustworthiness of the attester, producing attestation results. The relying party then applies its own appraisal policy to make an application-specific decision, such

as performing a privileged action or releasing confidential data to the attester. This process is illustrated by Figure 3.7.

We will assume that the verifier has already established trust and a secure communication channel with the relying party. RATS outlines how the trust between those two roles can be established [8, Section 7.1]

3.3.1 Chain of Trust

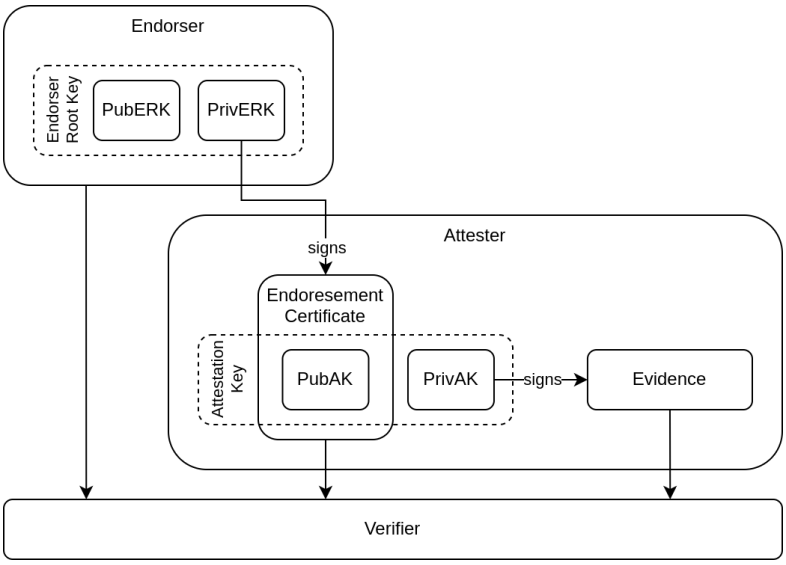

Figure 3.8: Remote attestation chain of trust.

Typically, a verifier comes to trust an attester indirectly by having an endorser, creating a chain of trust. This chain of trust is rooted at the endorser root key. The endorser provisions each attester with an attestation key, which is used to sign evidence. At the very least, the private part of the attestation key has to be stored in tamper-proof hardware, in order to prevent unauthorized access to the key. The endorser then issues an endorsement certificate which includes the public attestation key and signs the certificate using its own private endorsement root key.

The endorsement certificate represents the endorsement from the endorser that the attester can be trusted. Both the endorsement certificate and the public part of the endorser root key have to be provided to the verifier. Before starting its first verification, the verifier has to authenticate the endorsement certificate by validating its signature using the public endorser root key.

Evidence produced by the attester is signed by the attester using the attestation key. After receiving evidence from the attester, the verifier first verifies that the evidence was produced by the attester by validating its signature using the public attestation key, which is included in the endorsement certificate. Subsequently, the verifier confirms the integrity of the evidence using the validated signature.

After validating the authenticity and integrity of the evidence, the verifier appraises the evidence using reference values provided by the reference provider.

3.3.2 Secure Communication Channel between Attester and Relying Party

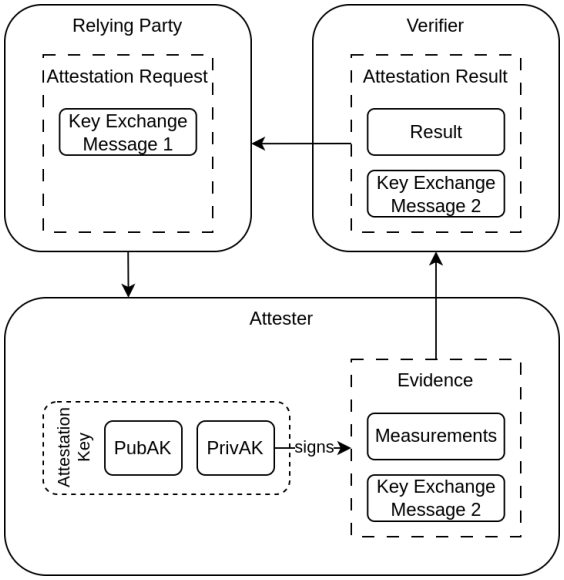

Figure 3.9: Integration of the Diffie-Hellman key exchange protocol into the remote attestation process.

The remote attestation process can also be used to establish a secure communication channel between the attester and the relying party. This can be achieved by integrating a key agreement protocol into the remote attestation process. This section will outline how a secure channel can be established between the attester and the relying party using the Diffie-Hellman key exchange as an example (see Section 2.1.2). We will assume that the prime number p and generator g are already specified by the protocol or are already agreed upon.

The relying party starts the attestation process, by requesting a verification of the attester. In the request the relying party includes the first key exchange message. The

attester then produces evidence which includes the second key exchange message. Because the evidence is signed by the attestation key it is part of the chain of trust, allowing the verifier to validate the authenticity and integrity of the key exchange message. After a successful verification the verifier includes the key exchange message in the attestation result. Now both the attester and the relying party have exchange their key exchange messages, which included pub_A and pub_B, allowing the derivation of a shared secret key K that can be used to establish a secure communication channel between both parties.

A man-in-the-middle attack between the attester and the verifier can easily be detected, because modification of the key exchange message would result in the verification of the integrity of the evidence to fail.

While the previous illustration is about the integration of the establishment of a secure communication channel into the remote attestation process, current ongoing work by Tschofenig et al. [42] proposes the reverse, integration of the remote attestation process into the already well established TLS protocol.

4 Trusted Distributed Computing Model

Section 3.1 described the traditional distributed computing model, where the service provider is trusted by the data owner, and defined a generalized layered model. This chapter will define a threat model, identify threats and issues of the Infrastructure and Platform layers when the service provider becomes untrusted. Subsequently, the new trusted distributed computing model will be defined, by integrating TEE technology (see Section 3.2) and the RATS framework (see Section 3.3) into the traditional distributed computing model. Finally, this chapter demonstrates this new model by looking at two case studies and evaluates the new model based on the threat model.

4.1 Threat Model

4.1.1 Infrastructure Layer

Threats from storage resources

Storage resources of the Infrastructure layer are typically protected by physically securing the location of the hardware and application transparent encryption. However, in the trusted distributed computing model the service provider is untrusted, and the application owner has to implement application level encryption, preventing the service provider gaining plain text access to confidential data.

Threats from networking resources

Networking resources offer means of communication. However, in a distributed computing environment, these communication channels are often untrusted as these resources are shared between multiple tenants. An attacker might gain unauthorized access to confidential data shared by an application by capturing data from the shared network resources. Applications sharing confidential data using untrusted communication channels need to implement authentication [24], authorization [45], and encryption in order to securely use these untrusted communication channels and not leak confidential data to a malicious attacker that has access to network resources. Authentication and encryption is usually implemented using the Transport Level Security (TLS) protocol [14], which utilizes a trusted third party (the certificate authority) for authentication and the

Diffie-Hellman key exchange protocol in order to establish a shared secret for encryption.

Typically, these implementations do not rely on services provided by the service provider and as such pose no issues for moving to an untrusted distributed computing model.

Threats from compute resources

Techniques for protecting data resting in storage and traversing networking resources largely are based on cryptographic primitives, such as encryption, one-way hash functions, and digital signatures. All of these primitives require some kind of cryptographic key (e.g. secret key for symmetrical encryption, private keys for asymmetrical encryption and singing) that need to be kept secret. Typically, these keys and the decrypted data are stored unprotected in the memory of the machines while in use and is therefore a prime target for attacks.

There are two types of physical attacks, related to the memory of a system, summarized by Weis [44]: **Direct Memory Access (DMA) Attacks** exploit the design of the x86 instruction set architecture that allow hardware subsystems to bypass the CPU and directly access the memory of the machine. Like the name implies, **Physical Memory Extraction (PME) Attacks** directly extract data from the physical system memory of a machine. Example for this type of attack are monitoring the memory bus of a machine, cold boot attacks, and exploiting special persistent, nonvolatile memory modules. While software and hardware based mitigations to DMA attacks exist, the firmware and software providing these mitigations are in control of the service provider. A malicious service provider could modify the firmware and software components to only pretend to mitigate those threats. In the past there were no countermeasures for PME attacks available (today confidential computing tries to address this issue 3.2).

One primary benefit that virtualization brings is isolation with the goal of shielding a VM from other VMs. Traditionally, the hypervisor is assumed to be trusted. As such it has been a high-privileged software component responsible for the virtualization of a VM's CPU, mapping of a VM's virtualized physical memory address space to the host machines physical memory address space, and intercepting and carrying out privileged operations invoked by a VM, in order to provide VM management tasks such as starting, stopping, suspending, restoring, and migrating VMs. But a malicious administrator or an attacker that gained access to hypervisor level privileges by exploiting vulnerabilities can completely monitor and modify resources available to another VM that may contain confidential data. Over the years there have been many vulnerabilities found in commonly used

hypervisors that break the isolation promises of hypervisors [32, 34]. We will refer to this kind of attacks as **Virtualization-based Attacks**.

Measures for protecting VM's from the hypervisor mostly separate the high privileged operations into a small component, that is either implemented in hardware or as a software component, that control the access to these operations and move the rest of the hypervisor into a less privileged execution mode [22, 40, 25, 28]. However, all of these mitigations are based on the trust that the party managing the hypervisor properly selects, configures, and patches the hypervisor. There are often no means for application owners to enforce or verify the selection, configuration, and update policies or and are also not able confirm that the hypervisor has not been tampered with.

Aside from physical threats, Weis also describes the threat of **Boot Integrity (BI) Attacks**. While the operating system of a machine can often be provided by the application owner, firmware, such as the BIOS, used in both physical and virtual environments providing hardware abstraction, are often controlled by the service provider. Consequently, BI attacks are applicable in physical and virtual environments. BI attacks exploit the high level of privilege and the fact, that firmware may be invisible to the operating system, in order to compromise security measures in higher levels of software. As such, establishing trust in the provided machines requires every piece of software that is executed during the lifetime (including the firmware) of the machines to be measured, verified, or isolated.

There have been two main approaches to prevent BI attacks:

The **Secure Boot** approach verifies the signatures of each software component loaded during the boot process of a machine. This requires the maintenance of a certificate which is used to verify the signatures of the firmware, bootloader, and kernel of the operating system.

But Secure Boot does not provide means to know what specific components loaded during the boot process, only that those components are verified using the provided certificate. **Measured Boot** aims to provide a trusted record of the boot process. Traditionally, this has been done using a Trusted Platform Module (TPM) which contains Platform Configuration Registries (PCRs) that are used to store measurements of loaded software components. These measurements are taken by various software components, such as the firmware, or bootloader. After the boot the PCRs are sealed, preventing the modification of the measurements, and signed by the TPM. Measured Boot then relies on a remote attestation process

(see Section 3.3), to recover the signed set of measurements, verify the signature, and evaluate whether the measurements follow a known policy.

Secure boot and Measured Boot rely on a trusted software component to verify and/or measure other software components required for the boot process. However, in an untrusted infrastructure environment tenants can not trust these components, as these can be modified by the service provider.

Resource	Threat	Mitigation	Issue
Storage Resources	Access of data resting in storage resources	Application level encryption of data resting in storage resources	
Network Resources	Access of data traversing network resources	Application level authentication and authorization, and encryption of data traversing network resources	
Compute Resources	DMA attacks	Software and Hardware based mitigations available	Lack of a trusted component that can verify that mitigation is working as intended
	PME attacks		No mitigation
	Virtualization-based attacks	Properly selected, configured, patched, and unmodified hypervisor	Lack of trusted component that can enforce or verify the selection, configuration, update policies, and integrity of the hypervisor
	BI attacks	Secure Boot & Measured Boot	Lack of a trusted component that can perform measurements and/or verification

Table 4.1: Overview of Infrastructure layer threats, typical mitigations in the traditional distributed computing model, and arising issues when moving to the trusted distributed computing model.

4.1.2 Platform Layer

Infrastructure management services typically have full access to Infrastructure layer resources. Therefore, the same threats of the Infrastructure layer apply here.

Security services offer application-level authentication, authorization, and/or encryption, support the process of implementing security into an application. However, a malicious service provider might modify or access those services in order to use another user's credentials (Spoofing), gain privileged access to resources and services (Elevation of Privilege), and/or gain plain text access to confidential data.

Monitoring and Logging services aggregating metrics and logs of applications in the Platform layer. While monitoring metrics usually do not contain sensitive information, the application logs can in some cases contain sensitive information.

Application orchestration services automate the process of preparing target execution environments, and installing, configuring, and maintaining applications inside the target environments.

A malicious service provider might misuse application orchestration services in order to:

- execute malicious code in the provided target environments.
- modify applications or their configurations, directly influencing the behavior of those applications.

Service Type	Issue
Infrastructure Management Services	Same as Infrastructure layer
Security Services	Spoofing, Elevation of Privilege, plain text access
Monitoring & Logging Services	Access to possibly sensitive logs
Application Orchestration Services	Execution of malicious code in target execution environment
	Modification of applications or their configurations

Table 4.2: Issues in the Platform layer when moving to an trusted distributed computing model.

4.2 Trust Model

The roles of the trusted distributed computing model are still present, with the difference that the service provider is no longer consider trusted. However, there are now three new roles present, based on the RATS framework, and the service provider is given additional responsibilities:

Hardware Manufacturer

The hardware manufacturer provides hardware to the service provider that form the base for the Infrastructure layer. The hardware has to be able to provide TEEs and the hardware manufacturer endorses the security of those TEEs. As such the hardware manufacturer takes on the endorser role defined by RATS (see Section 3.3). Even though, this model still applies when multiple TEE platforms and hardware manufacturers are present, for simplicity we will assume that only a single TEE platform and hardware manufacturer is present.

Service Provider
: In this model service providers not only provides Infrastructure layer resources and manages Platform layer services, but now utilize a TEE platform provided by the hardware manufacturer in order to offer the capability of creating TEEs and getting evidence for the integrity of the TEEs. These TEEs will be used by the application owner to execute applications in a trusted environment.

Reference Value Provider
: Reference value providers generate reference value of TEEs in advance, which is then used by the verifier in order to validate TEEs offered by the service provider.

Verifier Owner
: An entity that operates the verifier, a new component introduced in this model that is responsible for assessing the trustworthiness of TEEs provided by service providers.

Application Owner
: The responsibilities of application owners did not change. However, because in this model the service provider is untrusted, services of the Platform layer are also untrusted. As such, the application owner has to ensure that the security of the Application layer does not depend on services of the Platform layer.

In this model, the service provider is the only role that is not trusted by the data owner. Because every other role is trusted, a single entity or organization can take on multiple roles. However, the entity taking on the role of the service provider can not take on any other roles.

4.3 Architectural Overview

4.3.1 Verifier

The verifier is a required component of the trusted distributed computing model, responsible for verifying TEE integrity and is operated by the verifier. When a relying party, usually the application or data owner, requests the verification of an TEE, the verifier validates that the of evidence provided by the service provider is produced by the TEE platform using endorsements from the hardware manufacturer, and verifies the integrity of the TEE using the evidence and reference values provided by the reference value provider.

Software components involved in the creation process of TEEs, such as firmware components of the TEE platform or VM firmware, are still under control of the service provider. So in order for the verifier to validate the integrity of the TEE, evidence

produced by the TEE platform also has to include measurements of said software components that have to be compared to reference values in order to verify the integrity of those components.

The verifier is only responsible for verifying the integrity of TEEs as a whole and does not verify single applications inside TEEs. The verification of applications inside TEEs is only a side effect of verifying TEEs as a whole.

4.3.2 Infrastructure Layer

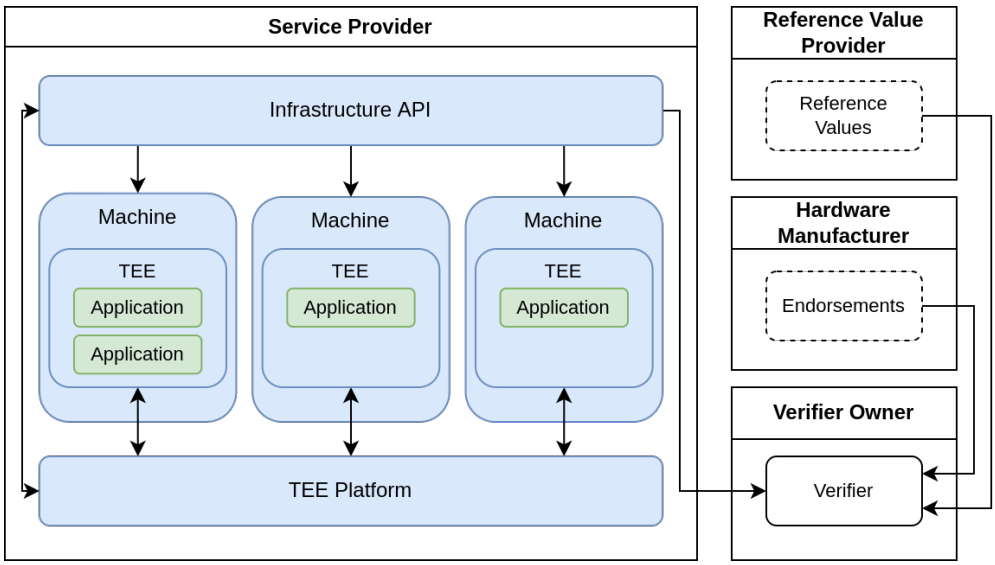

Figure 4.1: Overview of the infrastructure in the trusted distributed computing model.

As outlined in the threat model, protection of data resting in storage resources and traversing network resources has to be implemented at application level using authentication, authorization, and encryption.

Instead of directly running applications on provided (physical or virtual) machines, the data and application owner rely on TEEs to protect data that is currently in use by applications. In Section 3.2 we have seen two TEE models. On one hand, a service provider can provide machines capable of creating process-based TEEs inside those machines, on the other hand, a service provider can also provide VM-based TEEs, in which case the machine itself is the TEE.

The TEE platform has to be capable to produce evidence of software components that have direct access and/or control the access to the memory of a TEE, such as VM

firmware and firmware components of the TEE platform. While the TEE platform is operated by the service provider, the TEE platform is provided to the service provided and endorsed by the hardware manufacturer.

A secure channel is essential for data owners to supply applications inside TEEs with confidential data. Section 3.3.2 outlined the basic process of establishing a secure communication channel between the attester and the relying party during the remote attestation process.

4.3.3 Platform Layer

Like in the traditional distributed computing model, the Platform layer builds on top of the Infrastructure layer. Therefore, this section assumes the presence of a TEE platform that can be used to securely deploy applications and provide the application with confidential data.

Because in this model the service provider is untrusted, there are three possible solutions to the issues of the services typically located in the Platform layer:

- Moving these services to the Application layer or directly implementing the tasks that these services take on into applications.
- Management of the service by a trusted party that is not the application owner (otherwise it would mean that the service is in the application layer).
- Splitting the service into a privileged and unprivileged part. The unprivileged part can still be operated by the service provider, while the privileged part has to be managed by the application owner.

The first option can be applied to all services of the Platform layer, as such this section will focus on the evaluation of each previously defined service type based on the latter two options.

Infrastructure Management Services do not need to be modified and can still be managed by the service provider, assuming applications are adequately shielded from Infrastructure layer resources as described in the previous section.

Security Services can not be managed by an untrusted party, as application-level security depends on those services. Management of identities, administration of authentication policies, and encryption and decryption of data are all privileged tasks. Therefore, splitting security services into unprivileged and privileged parts is not feasible and these services have to managed by a trusted party.

Monitoring and Logging Services can be managed by a service provider, assuming that application logs do not include sensitive information or are protected (e.g. encrypting or obfuscating logs).

Application Orchestration Services are a more complex topic. Because the TEE platform is still under control of the service provider, software responsible for managing TEEs are still untrusted (e.g. hypervisor, OS managing process-based TEEs). Applications and application configurations can also be modified by an attacker during the deployment of applications into TEEs. Therefore, TEEs, applications, and application configurations have to be verified before supplying applications with confidential data.

There are two ways on how applications can be deployed into TEEs:

The first option is to create an application package that can be directly deployed by the TEE platform. For example the application owner might package an application including its (non-confidential) configuration directly into an VM image that can be deployed by the service provider. The application owner has to provide reference values in the form of measurements of the VM, in order for the verifier to verify the integrity of the VM including the applications inside. This option moves the installation and configuration of applications into the application packaging process, and therefore requires a more complex packaging solution. However, it also allows the combined verification of TEEs, applications, and application configurations.

The second option is to first create a generic application-agnostic TEE and subsequently deploy the application including its (non-confidential) configuration into the TEE after verifying its integrity. For example a service provider might provide a curated list of generic VM images that can be deployed by tenants as VM-based TEEs. A reference value provider has to deploy such a VM image, check its content for malicious software, and create measurements of the VM beforehand, producing reference values. After a VM-based TEE is created, the verifier has to verify the integrity of the VM-based TEE using the reference values, after which the application owner installs, configures, and executes applications in the verified VM. This option decouples the creation of TEEs from the installation, configuration, and execution of applications, allowing another reference value provider to maintain reference values for TEEs. However, this also decouples the verification of TEEs from the verification of applications and their configurations, requiring two separate verifications.

While in both cases the service provider creates TEEs, the service provider is not involved in the installation, configuration, and execution of applications in the

provided TEEs. Both options also require a secure way to supply the applications with confidential data, which can again be achieved by establishing a secure communication channel during the remote attestation process.

4.3.4 Application Layer

The threat model above already discussed how application level authentication, authorization, and encryption of confidential data is crucial to protect data resting in storage resources and traversing network resources. As the service provider is not trusted, Infrastructure and Platform layer protection methods can not be used, and the application owner has to implement these protections into the Application layer.

4.4 Requirements

This section summarizes the changes made to the traditional distributed computing model and defines requirements for the trusted distributed computing model.

The trusted distributed computing model is largely based on concepts of the confidential computing model and requires a hardware manufacturer to provide:

- R1 a TEE platform, capable of creating TEEs and generating evidence, based on hardware mechanisms.
- R2 endorsements that vouch for the TEE platform's capability to securely generate evidence and protect the confidentiality and integrity of TEEs.

There are two requirements for the offerings of a service provider. The service provider has to provide:

- R3 the ability to create TEEs using the TEE platform provided by the hardware manufacturer.
- R4 the raw evidence produced and signed by the TEE platform that includes:
  - R4.1 measurements of the content of the TEE.
  - R4.2 measurements of software components that have direct access or the access to the memory TEEs.

The latter requirement enables the verifier to verify TEEs, preventing the service provider to tamper with provided TEEs in order to break the confidentiality and integrity guarantees. This leads to the requirements for the verifier. The verifier has to verify

R5 the integrity of provided TEEs using measurements provided by the service provider and reference values from the reference value provider.

R6 the integrity of the TEE platform by verifying:

R6.1 the authenticity of the evidence using the evidence's signature and endorsements from the hardware manufacturer.

R6.2 the integrity of software components that have direct access or control the access to the memory of TEEs using measurements of those components and reference values from a reference value provider.

While TEEs protect data currently in use by applications, data resting in storage resources and traversing network resources also have to protected. The application management process also has to be secured, preventing the service provider to execute malicious applications inside TEEs and modifying applications before they are deployed into TEEs. This results in the following requirements for the application owner. The application owner has to:

R7 implement security (authentication, authorization, and encryption) in the Application layer, not relying on security services provided by the Platform layer.

R8 protect application logs by removing, encrypting, or obfuscating confidential information from the logs.

R9 deploy applications inside TEEs. This includes:

R9.1 providing reference values for applications and their configuration.

R9.2 installing, configuring, and executing applications inside TEEs.

R9.3 securely supplying applications with confidential data only after verification of applications and their surrounding TEE (e.g. by establishing a secure communication channel during the remote attestation process).

4.5 Evaluation

This section evaluates the requirements of the trusted distributed computing model based on the threat model defined in section 4.1.

Infrastructure layer threats from storage and network resources are addressed in the trusted distributed computing model by R7. By implementing authentication, authorization, and encryption at application level, an attacker that gained access to storage and network resources or the untrusted service provider providing these resources, still do not have plain text access to confidential data.

Threats from compute resources of the Infrastructure layer are addressed by R1-R6. The issue of traditional mitigations of these threats was the lack of a trusted component that can verify the integrity of components that implement the mitigations. R3 and R4 require the service provider to offer the capability to create TEEs using and get evidence produced and signed by a TEE platform. The TEE platform is provided and endorsed by the hardware manufacturer (R1 and R2). Section 3.2 described how TEEs protect memory confidentiality and integrity using memory encryption and access control.

Physical access to a machine's memory (DMA and PME attacks) are prevented as data leaving the CPU is encrypted before storing it in memory. As such, an attacker with direct access to a machine's memory still does not have plain text access to the data.

Virtualization-based attacks are also mitigated by the TEE platform. VM-based TEEs split of the memory access control and isolation part of the hypervisor of into the TEE platform. The hypervisor is still provided with interfaces in order to manage VMs (e.g. starting, stopping, suspending), but the isolation guarantee is provided by the TEE platform. And while the hypervisor is also tasked with managing VM memory (e.g. attaching and releasing), the hypervisor can not read the memory as it is encrypted.

Measurements produced by the TEE platform can also be used for the measured boot process mitigating BI attacks against VM-based TEEs. Ongoing work in QEMU, an open source emulator that can facilitate hardware-assisted virtualization in order to run VMs, the open virtual machine firmware (OVMF), and the Linux kernel enable the verification of all components involved in the boot process of an AMD SEV based VM (see Section 3.2.3). The basic boot process works as follows: When a VM is started, the AMD-SP measures the VM's firmware (OVMF), which in turn measures and verifies the integrity of the Linux kernel. After the VM has booted, the Linux kernel is supplied with an attestation report that contains a measurement of OVMF, which are relayed to the verifier in order to verify OVMF's integrity. The attestation report also contains evidence about the integrity of SEV's firmware components which together with the measured boot process fulfill requirements R4, allowing a verifier that supports the verification of SEV's attestation report to fulfill R5 and R6.

The TEE creation capability of process-based TEE platforms can often be passed through to VMs (e.g. in the case of Intel SGX), in which case the VM itself and the hypervisor are not considered trusted. Again, isolation is provided by the TEE platform (R3), and not the hypervisor. In this model, compromising the VM on which the process-based TEE is deployed only threatens the availability of the application, and not the confidentiality of the application's data. TEE platforms based on the process-based TEE model, still need to provide measurements of software components of the TEE platform and the TEEs (R4), allowing a verifier to verify the integrity of the TEE platform (R6) and TEEs (R5).

R7 requires the implementation of security mechanisms in the Application layer. These mechanisms can not be based on security services provided by the Platform layer, as this would allow spoofing and elevation of privilege attacks, or plain text access to confidential data from the untrusted service provider. While R8 does not require moving monitoring and logging services into the application layer, it does require the Application layer to protect sensitive information in application logs. Infrastructure management services can still be part of the Platform layer, as confidential is already protected from Infrastructure layer resources by R1-R7.

Application orchestration however, still relies on management of target environments for applications to be managed by untrusted software components. That is VM-based TEEs managed by a hypervisor or process-based TEEs managed by the surrounding host (or guest) OS. R9 requires that the rest of the application orchestration process, that is installation, configuration, and execution of applications inside provided target environments, is managed by the application owner. It also requires the verification of both TEEs and applications before supplying applications with confidential data. Requiring the verification of applications enables the detection of modification of applications or their configurations, and requiring the application owner to manage the execution of applications inside verified TEEs, prevents the service provider to execute malicious code.

On the one hand, all R7-R9 remove the service provider as a trusted party, but on the other hand they significantly increase the responsibilities of application owners and the size of the Application layer.

4.6 Case Studies

4.6.1 Case Study: Constellation

Constellation is a project maintained by Edgeless Systems[1]. By extending Kubernetes, Constellation provides a platform base that includes infrastructure management and application orchestration services (see Section 3.1.5) on top of the Infrastructure layer provided by an untrusted cloud provider. Its contribution is the protection of the whole Kubernetes cluster from underlying Infrastructure layer resources by utilizing VM-based TEEs, and providing transparent network and storage encryption.

[1] `https://docs.edgeless.systems/constellation/`

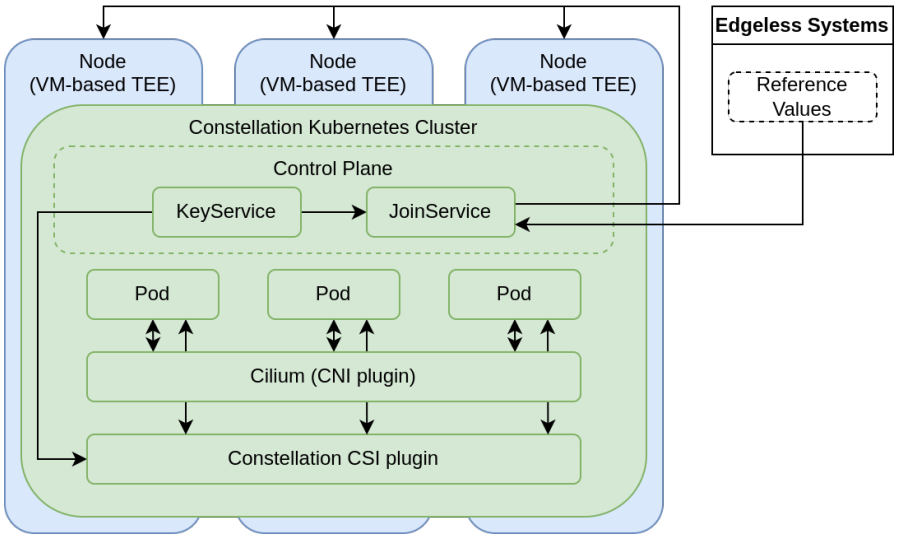

Figure 4.2: Constellation Kubernetes overview.

In order to protect data currently in use by control plane components and application pods, Constellation runs the whole cluster on VM-based TEEs. The JoinService, a new component of the control plane, is responsible for verifying new nodes joining the cluster. After a node that is supposed to run control plane components is verified and joins the cluster, it is supplied by the JoinService with encryption keys supplied by the KeyService. These keys are then used to encrypt etcd data that is stored on the node, protecting control plane data at rest.

In this architecture, the JoinService correlates to the verifier introduced previously. It uses reference values provided by Edgeless Systems in order to verify joining nodes. The KeyService manages cryptographic keys used for encryption and decryption and is therefore a security service.

Protection of control plane and application data in transit is provided by cilium, the CNI plugin chosen by constellation, which encrypts data exchanged between pods without the need for applications running inside pods to be modified. Persistent application data at rest is shielded by the CSI plugin provided by Constellation, which encrypts and decrypts persistent volumes using keys provided by the KeyService without needing modification of the applications inside pods. Both the CNI and the CSI plugin are implementations of infrastructure and security services

Constellation currently only supports AMD SEV, making AMD take on the role of hardware manufacturer that provides and endorses the TEE platform (R1 and R2). However, the currently supported cloud providers (Azure, GCP, AWS) do not fully support the trusted distributed computing model at the time of writing. For exam-

ple AWS currently does not provide the capability of providing SEV based VMs, not meeting R3. While Azure and GCP provide SEV based VMs, Azure currently does not provide reviewable VM firmware, not meeting R4.2, and GCP does not provide evidence produced by the SEV platform, not meeting R4.

Because the cloud providers do not meet R3 and R4, the verifier in the Constellation architecture (JoinService) can not fulfill R5 and R6. On the one hand, if the cloud provider does not meet R3, there are no TEEs to be verified, making complying with R5 impossible. On the other hand, if the cloud provider does not meet R4, the verifier can not fully verify the integrity of the TEE platform or TEE, preventing the verifier to satisfy R6.

While Edgeless Systems provides application owners with tools to create and manage a Constellation Kubernetes cluster, the cluster still has to be administered by the application owner, putting the whole cluster into the Application layer. In doing so Constellation meets requirements R7 and R9, because security services (KeyService, CNI, and CSI plugin) and application orchestration services (natively included in Kubernetes) are now managed by the application owner.

4.6.2 Case Study: Confidential Containers

Confidential Containers is another project, trying to implement the untrusted distributed computing model into the Kubernetes architecture[2]. The key difference, is that while Constellation applies confidentiality at the node level, Confidential Containers applies confidentiality at a pod level. By running containers inside TEEs (both VM-based and process-based) and not the whole cluster, Confidential Containers goal is to separate the trust model of the Kubernetes cluster from the applications deployed in the cluster. The project is still in a very early development stage, as such information provided in this section is subject to change.

The current Kubernetes architecture puts the container runtime in charge of managing pods and containers inside the pod (see Section 3.1.5). Confidential Containers introduces a new component into the pod: the enclave agent. The enclave agent collects claims generated by the TEE platform, performs the remote attestation process, receives confidential configuration and data, pulls container images from the container image registry, and manages the lifecycle of containers inside the pod. In the case of VM-based TEEs, a pod is represented by a single VM, which includes the enclave agent as a process that is started at boot and the containers are started inside the VM. For process-based TEEs, the enclave agent is a separate process-based TEE, that creates

[2] `https://github.com/confidential-containers/documentation`

distinct process-based TEEs for each container on the node itself. In this case study we will illustrate the Confidential Container architecture only using VM-based TEEs.

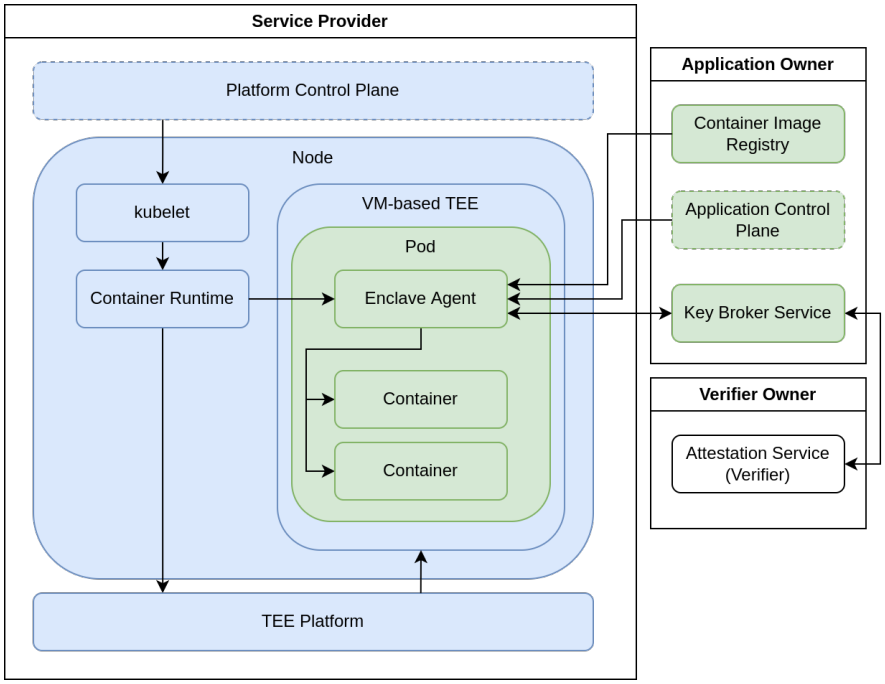

Figure 4.3: Confidential Containers application orchestration overview.

While the agent only acts upon request from a single API source, traditionally the Kubernetes control plane, Confidential Containers is currently working on splitting the Kubernetes control plane into a trusted and an untrusted part. The platform control plane (untrusted) would be responsible for unprivileged tasks, such as infrastructure and TEE management. On the other hand, the application control plane (trusted) performs privileged tasks, including container management inside TEEs provided by the platform control plane. The split of the control plane is enforced by the enclave agent, by blocking privileged actions originating from the platform control plane, such as creating a malicious container in a pod, and establishing a secure communication channel to the application control plane.

The architecture builds upon the assumption that confidential data is provided to the pod using persistent volumes managed by a CSI plugin, but stored in an encrypted form. Decryption keys are provided to pods by the key broker service, which correlates to the relying party of the RATS framework. It receives evidence from the enclave agent, relays the evidence to the verifier (called attestation service in the Confidential

Containers architecture) for verification, applies appraisal policies on the returned attestation results, and releases keys to the enclave agent. During the remote attestation process, a secure communication channel between the key broker service and the enclave agent is established, which is used for the secure release of keys to the enclave agent. Besides data decryption keys the key broker service also sends communication keys to the enclave agent that are used to establish a secure communication to other components, such as the application control plane.

Currently, it is still not clear, how the ConfigMap and Secret resources of the traditional Kubernetes architecture fit into this new architecture, which is why the following illustration of the application orchestration process does not include ConfigMaps and Secrets.

1. Upon request of a tenant, the platform control plane requests the kubelet of a node to create a pod and provides the kubelet with the specification of the pod. This specification mainly includes container images, storage configuration, and network configuration.
2. Kubelet calls the container runtime to create the pod and configure its storage and network.
3. The container runtime in turn calls the TEE platform to create a VM-based TEE using a predefined VM image that includes the enclave agent.
4. During the boot of the VM the TEE platform produces signed evidence that is then passed to the enclave agent after the VM has fully booted. The container runtime also passes the pod specifications to the enclave agent.
5. The enclave agent requests the release of keys from the key broker service and includes the evidence in the request.
6. The key broker service relays the evidence to the attestation service and receives an attestation result, upon which the key broker service applies its own appraisal policy.
7. If the attestation was successful, the key broker service releases keys and a pod specification policy to the enclave agent. The policy can for example include storage and network configuration, a container image whitelist, a certificate which can be used to validate the authenticity of container images.
8. The enclave agent then compares the pod specification it received from the container runtime to the policy. If the specification passes the policy, the enclave agent pulls the specified container images and validates the signature using the certificate included in the policy.

9. Only after passing the policy the enclave agent creates the containers inside the VM and provides them with decryption keys.

Note that the enclave agent is only trusted, because it is included in the VM image which is verified by the verifier. Modification of the enclave agent (or any other software in the VM image) would fail the verification and lead to the key broker service not releasing keys to the enclave agent.

5 Conclusion

The aim of this thesis was the introduction of a new trusted distributed computing model that protects the confidentiality of tenants data. The resulting model is based on the integration of confidential computing technologies and remote attestation procedures into the traditional distributed computing model. The defined threat model identified threats and issues that arise when service providers of the traditional model become untrusted and the evaluation of the trusted model upon this threat model has shown how trusted execution environments mitigate these threats and solve issues in existing mitigations.

The split of trust however comes with trade-offs. The traditional model allowed tenants to shift more and more responsibilities to the service provider, reducing the complexity of applications. The trusted model reduces this benefit by giving responsibilities that could lead to the compromise of the confidentiality of data back to the application.

We have also seen limitations that are currently still present in confidential computing technologies that are mainly concerned with performance limitations. These performance limitations diminish the cost-efficiency of distributed computing systems and in some use cases not acceptable.

Furthermore, confidential computing technologies are still a relatively new, and their integrations into other software such as hypervisors, firmware, operating systems, and into the remote attestation process are still in early stages, requiring further research and testing.

While cloud providers are eager to adopt this new technology and are already offering services based on confidential computing technologies, the two case studies have shown that both the Infrastructure layer and the Platform layer are currently still not fulfilling all requirements of the trusted distributed computing model.

Nevertheless, confidential computing provides very promising hardware-based primitives that could allow a new generation of trustworthy distributed computing systems. Further research will tell whether current limitations and implementation challenges can be solved while preserving the confidentiality promises.

Bibliography

[1] Ayaz Akram et al. “Performance Analysis of Scientific Computing Workloads on General Purpose TEEs”. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021, pp. 1066–1076. DOI: 10.1109/IPDPS49936.2021.00115.

[2] Ayaz Akram et al. “SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems”. In: *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. 2022, pp. 121–132. DOI: 10.1109/SEED55351.2022.00018.

[3] AMD. *AMD Memory Encryption*. Version 9.

[4] AMD. *Protecting VM Register State With SEV-ES*.

[5] AMD. *Strengthening VM isolation with integrity protection and more*. Version 1.54. Jan. 2020.

[6] Raad Bahmani et al. “{CURE}: A Security Architecture with {CUstomizable} and Resilient Enclaves”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 1073–1090.

[7] Mihir Bellare, Alexandra Boldyreva, and Adam O’Neill. “Deterministic and efficiently searchable encryption”. In: *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*. Springer. 2007, pp. 535–552.

[8] H. Birkholz et al. *Remote ATtestation procedureS (RATS) Architecture*. RFC 9334. RFC Editor, Jan. 2023. DOI: 10.17487/RFC9334. URL: https://www.rfc-editor.org/info/rfc9334.

[9] Jo Van Bulck et al. “Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution”. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1041–1056. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck.

[10] Tianshi Chen et al. "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning". In: *SIGARCH Comput. Archit. News* 42.1 (Feb. 2014), pp. 269–284. ISSN: 0163-5964. DOI: 10.1145/2654822.2541967. URL: https://doi.org/10.1145/2654822.2541967.

[11] Dave Clarke, Michael Richmond, and James Noble. "Saving the World from Bad Beans: Deployment-Time Confinement Checking". In: *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications*. OOPSLA '03. Anaheim, California, USA: Association for Computing Machinery, 2003, pp. 374–387. ISBN: 1581137125. DOI: 10.1145/949305.949339. URL: https://doi.org/10.1145/949305.949339.

[12] Confidential Computing Consortium. *A Technical Analysis of Confidential Computing*. 2022. URL: https://confidentialcomputing.io/ccc-a-technical-analysis-of-confidential-computing-v1-3_updated_november_2022/.

[13] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086. 2016. URL: https://eprint.iacr.org/2016/086.

[14] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. http://www.rfc-editor.org/rfc/rfc5246.txt. RFC Editor, Aug. 2008. URL: http://www.rfc-editor.org/rfc/rfc5246.txt.

[15] Shufan Fei et al. "Security Vulnerabilities of SGX and Countermeasures: A Survey". In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3456631. URL: https://doi.org/10.1145/3456631.

[16] Ian Foster, Carl Kesselman, and Steven Tuecke. "The anatomy of the grid". In: *Lecture Notes in Computer Science* 2150.2001 (2001), pp. 1–28.

[17] Olaf Grote, Andreas Ahrens, and César Benavente-Peces. "A Review of Post-quantum Cryptography and Crypto-agility Strategies". In: *2019 International Interdisciplinary PhD Workshop (IIPhDW)*. 2019, pp. 115–120. DOI: 10.1109/IIPHDW.2019.8755433.

[18] Michael Gruhn and Tilo Müller. "On the Practicability of Cold Boot Attacks". In: *2013 International Conference on Availability, Reliability and Security*. 2013, pp. 390–397. DOI: 10.1109/ARES.2013.52.

[19] Mohammed Faez Al-Jaberi and Anazida Zainal. "Data integrity and privacy model in cloud computing". In: *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*. 2014, pp. 280–284. DOI: 10.1109/ISBAST.2014.7013135.

[20] Jianyu Jiang et al. "CRONUS: Fault-isolated, Secure and High-performance Heterogeneous Computing for Trusted Execution Environment". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, pp. 124–143. DOI: 10.1109/MICRO56248.2022.00019.

[21] Jianyu Jiang et al. "CRONUS: Fault-isolated, Secure and High-performance Heterogeneous Computing for Trusted Execution Environment". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, pp. 124–143. DOI: 10.1109/MICRO56248.2022.00019.

[22] Seongwook Jin et al. "Architectural Support for Secure Virtualization under a Vulnerable Hypervisor". In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-44. Porto Alegre, Brazil: Association for Computing Machinery, 2011, pp. 272–283. ISBN: 9781450310536. DOI: 10.1145/2155620.2155652. URL: https://doi.org/10.1145/2155620.2155652.

[23] Ruriko Kudo et al. "Integrity Protection for Kubernetes Resource Based on Digital Signature". In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. 2021, pp. 288–296. DOI: 10.1109/CLOUD53861.2021.00042.

[24] Butler Lampson et al. "Authentication in Distributed Systems: Theory and Practice". In: *ACM Trans. Comput. Syst.* 10.4 (Nov. 1992), pp. 265–310. ISSN: 0734-2071. DOI: 10.1145/138873.138874. URL: https://doi.org/10.1145/138873.138874.

[25] Shih-Wei Li, John S Koh, and Jason Nieh. "Protecting cloud virtual machines from hypervisor and host operating system exploits". In: *Proceedings of the 28th USENIX Security Symposium*. 2019.

[26] Xingwei Liang and Yoohwan Kim. "A Survey on Security Attacks and Solutions in the IoT Network". In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. 2021, pp. 0853–0859. DOI: 10.1109/CCWC51732.2021.9376174.

[27] ARM Limited. *Arm Confidential Compute Architecture Software Stack Guide*. Version r1p0. Sept. 13, 2022. URL: https://documentation-service.arm.com/static/6320744be60c8274af98e79a.

[28] Zeyu Mi et al. "(Mostly) Exitless VM protection from untrusted hypervisor through disaggregated nested virtualization". In: *Proceedings of the 29th USENIX Conference on Security Symposium*. 2020, pp. 1695–1712.

[29] C Neuman. *Scale in Distributed Systems. In Readings in Dist. Comp. Syst.* 1994.

[30] Monique Ogburn, Claude Turner, and Pushkar Dahal. "Homomorphic Encryption". In: *Procedia Computer Science* 20 (2013). Complex Adaptive Systems, pp. 502–509. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.09.310. URL: https://www.sciencedirect.com/science/article/pii/S1877050913011101.

[31] Claus Pahl. "Containerization and the PaaS Cloud". In: *IEEE Cloud Computing* 2.3 (2015), pp. 24–31. DOI: 10.1109/MCC.2015.51.

[32] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. "Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers". In: *Proceedings of the 2013 International Workshop on Security in Cloud Computing*. Cloud Computing '13. Hangzhou, China: Association for Computing Machinery, 2013, pp. 3–10. ISBN: 9781450320672. DOI: 10.1145/2484402.2484406. URL: https://doi.org/10.1145/2484402.2484406.

[33] Tim Grance Peter Mell. *The NIST Definition of Cloud Computing*. URL: https://doi.org/10.6028/NIST.SP.800-145.

[34] Jenni Susan Reuben. "A survey on virtual machine security". In: *Helsinki University of Technology* 2.36 (2007).

[35] Thomas Ristenpart et al. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 199–212. ISBN: 9781605588940. DOI: 10.1145/1653662.1653687. URL: https://doi.org/10.1145/1653662.1653687.

[36] Luis Rodero-Merino et al. "Building safe PaaS clouds: A survey on security in multitenant software platforms". In: *Computers & Security* 31.1 (2012), pp. 96–108. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2011.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0167404811001313.

[37] Fernand Lone Sang, Vincent Nicomette, and Yves Deswarte. "I/O Attacks in Intel PC-based Architectures and Countermeasures". In: *2011 First SysSec Workshop*. 2011, pp. 19–26. DOI: 10.1109/SysSec.2011.10.

[38] Felix Schuster. *Confidential Computing - How to process data securely on third-party infrastructure*. URL: https://content.edgeless.systems/hubfs/Confidential%20Computing%20Whitepaper.pdf.

[39] Maarten van Steen and Andrew S. Tannenbaum. *Distributed Systems*. 3rd ed. 2017.

[40] Jakub Szefer and Ruby B. Lee. "Architectural Support for Hypervisor-Secure Virtualization". In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVII. London, England, UK: Association for Computing Machinery, 2012, pp. 437–450. ISBN: 9781450307598. DOI: 10.1145/2150976.2151022. URL: https://doi.org/10.1145/2150976.2151022.

[41] Chia-Che Tsai et al. "Cooperation and Security Isolation of Library OSes for Multi-Process Applications". In: *Proceedings of the Ninth European Conference on Computer Systems*. EuroSys '14. Amsterdam, The Netherlands: Association for Computing Machinery, 2014. ISBN: 9781450327046. DOI: 10.1145/2592798.2592812. URL: https://doi.org/10.1145/2592798.2592812.

[42] Hannes Tschofenig et al. *Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Internet-Draft draft-fossati-tls-attestation-02. Work in Progress. Internet Engineering Task Force, Oct. 2022. 20 pp. URL: https://datatracker.ietf.org/doc/draft-fossati-tls-attestation/02/.

[43] Ieuan Walker, Chaminda Hewage, and Ambikesh Jayal. "Provable Data Possession (PDP) and Proofs of Retrievability (POR) of Current Big User Data". In: *SN Computer Science* 3.1 (Nov. 2021), p. 83. ISSN: 2661-8907. DOI: 10.1007/s42979-021-00968-z. URL: https://doi.org/10.1007/s42979-021-00968-z.

[44] Stephen Weis. "Protecting data in-use from firmware and physical attacks". In: *Black Hat* (2014).

[45] Thomas YC Woo and Simon S Lam. "Authorization in distributed systems: a formal approach." In: *IEEE Symposium on Security and Privacy*. Citeseer. 1992, pp. 33–50.

[46] Pan Yang, Naixue Xiong, and Jingli Ren. "Data Security and Privacy Protection for Cloud Storage: A Survey". In: *IEEE Access* 8 (2020), pp. 131723–131740. DOI: 10.1109/ACCESS.2020.3009876.

[47] Faheem Zafar et al. "A survey of cloud computing data integrity schemes: Design challenges, taxonomy and future trends". In: *Computers & Security* 65 (2017), pp. 29–49. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2016.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0167404816301377.