Karlsruhe Institute of Technology

Ensuring Confidentiality in a Distributed Computing Environment

Bachelor's Thesis of

Wenzhe Vincent Cui

at the Department of Informatics
SCC – Steinbuch Centre for Computing

Reviewer:	Prof. Dr. Achim Streit
Second reviewer:	Prof. Dr. Bernhard Neumair
Advisor:	Peter Krauss

30. November 2022 – 30. March 2023

Abstract

In the current computing landscape, distributed computing systems, primarily 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 services comes with benefits such as higher availability, reducing the complexity of applications, and cost-efficiency, on the other hand, these benefits come with the cost of fully trusting the provider of the resources and services with potentially confidential data. However, as the demand from consumers and governments for data privacy increases (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 providers of resources and services become untrusted by utilizing hardware-based trusted execution environments (TEEs). This thesis researches the current state of confidential computing technologies, remote attestation procedures that allow tenants to assess the trustworthiness of TEEs, and introduces a new *trusted distributed computing model* that integrates these two concepts into the traditional distributed computing 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 mainly based on a layered architecture where each layer's security depends on the security of the layers beneath it. The lowest layer is the hardware appliances that form the foundation of distributed computing systems, and the highest layer is the collection of user 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 primarily 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 that tenants want to execute. Traditionally, this protection only isolates 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 and employees 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 security functionality, such as authentication, authorization, 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 their own key management service (KMS) or service provider offered hardware security module (HSM). While this solution protects cryptographic keys from the service provider, the 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 [22, 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 TEEs 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 protect applications and confidential data, tenants have to be able to verify that applications are running inside TEEs and that the service provider has not modified the TEE platform. 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 indented for remote attestation purposes.

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 for 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 unauthorized parties. In this background section, we will 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 – the secret key – is used for encryption and decryption. The key has to be securely distributed and stored 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 fast and efficient. However, one of the significant drawbacks of symmetric cryptography is the issue of key management. Securely storing and distributing keys to entities and components that require access to encrypted data is challenging. Furthermore, attackers that gain access to a key can quickly gain plain-text access to encrypted confidential data.

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

Digital signatures are a widely used application of asymmetric cryptography. A digital signature is a cryptographic primitive that verifies a dataset's integrity and authenticity. The data owner uses their private key to generate a unique digital signature cryptographically bound to the dataset's content. When the owner sends the dataset together with the signature to another party, the receiver can use the owner's public key to verify the signature's authenticity and the dataset's integrity. A successfully validated signature gives the recipient confidence that the data owner sent the dataset and that the dataset has not been modified during transit.

Asymmetric cryptography is slower than symmetric cryptography due to the complexity of the mathematical algorithms involved. One of the significant benefits of asymmetric cryptography is that it eliminates the need for secret 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, has to be stored securely.

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 the encryption and decryption of messages. This section illustrates a key agreement process using the popular 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: 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 $K = (pub_A)^B \mod p = (pub_B)^A \mod p = g^{AB} \mod p$
5. Alice and Bob can then use the secret key K to encrypt and decrypt messages sent 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. Suppose we assume that the prime number p and generator g are already specified in a specific protocol or have been pre-established. In that case, Alice and Bob have to exchange exactly two messages, pub_A and pub_B.

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.

Not only does the attacker gain plain-text access to the messages, but 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 are sent by Alice, although the attacker might have sent them.

To mitigate man-in-the-middle attacks, either Alice or Bob must authenticate the other party. This can be achieved by utilizing digital signatures. For example, Bob uses his own private key to sign the message that includes pub_B and sends the message and the signature to Alice. Alice then uses Bob's public key to verify the messages authenticity and integrity.

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 [29] as resource sharing, transparent distribution, openness, and scalability. Distributed computing facilitates distributed system principles in order to group a set of resources (possibly at a geographical distance) and provide tenants with a single, coherent view of these resources. In general, distributed computing systems allow users to share, manage 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 resources to 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 that offer means of discovering, managing access to, and using remote resources. Because those resources are not subject to centralized control, these protocols and interfaces must be standardized and open to enable interoperability [13].

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 five layers of the Grid computing model defined by Foster, Kesselman, and Tuecke [13] are as follows:

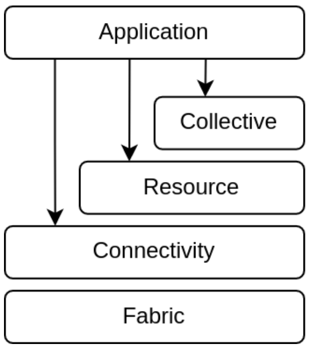

Figure 3.1: The layered Grid architecture.

Fabric Layer

The fabric layer provides the resources which are shared by the Grid system. It offers local, resource-specific operations called by sharing processes at higher layers. These operations provide resource inquiry and management capabilities.

Connectivity

This layer specifies core communication protocols, enabling the exchange of data between fabric layer resources. Commonly includes transport, routing, naming, authentication, and authorization 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 facilitates fabric layer functions to access and control resources. These protocols are focused on single resources, meaning they are not concerned with coordinating multiple resources.

Collective

While the resource layer provides interactions with a single resource, the collective layer focuses on interactions with a collection of resources. Being the top of the hourglass allows this layer to provide a wide variety of behaviors, such as discovering, co-allocating, scheduling, brokering, monitoring, and replicating resources.

Applications

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

3.1.2 Cloud Computing

Cloud Computing systems typically have centralized control and facilitate open and proprietary protocols and interfaces to provide on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured services. *The NIST Definition of Cloud Computing* [24] 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 neither manages nor controls the underlying infrastructure but can deploy and run arbitrary software on the provided infrastructure, including system software such as operating systems.

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. The consumer can run arbitrary applications supported by the platform but neither manages nor controls the provided platform services.

Software as a Service (SaaS)
: The SaaS service model again adds another layer of abstraction. Compared to the PaaS model, the consumer can not run arbitrary applications but only a provider-specific selection of applications. Furthermore, the Cloud provider controls and manages the applications, only allowing customers to customize a limited set of user-specific configurations.

These service models can be seen as hierarchical layers providing various resources and service, which customers can utilize to develop and deploy applications.

3.1.3 Trust Model

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

Service Provider
: Generally, 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 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 is being used or manipulated by an application. They are concerned about the confidentiality of their data. There may be mutual distrust between data owners if there are multiple 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, three virtual organizations *A*, *B*, and *C*, share resources and services. Assuming that virtual organization *A* owns dataset *D* but all of *A*'s resources are currently occupied. Therefore, *A* wants to use resources from virtual organization *B* 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 the application owner and service providers. Therefore, both roles can be taken on by the same party.

3.1.4 Architectural Overview

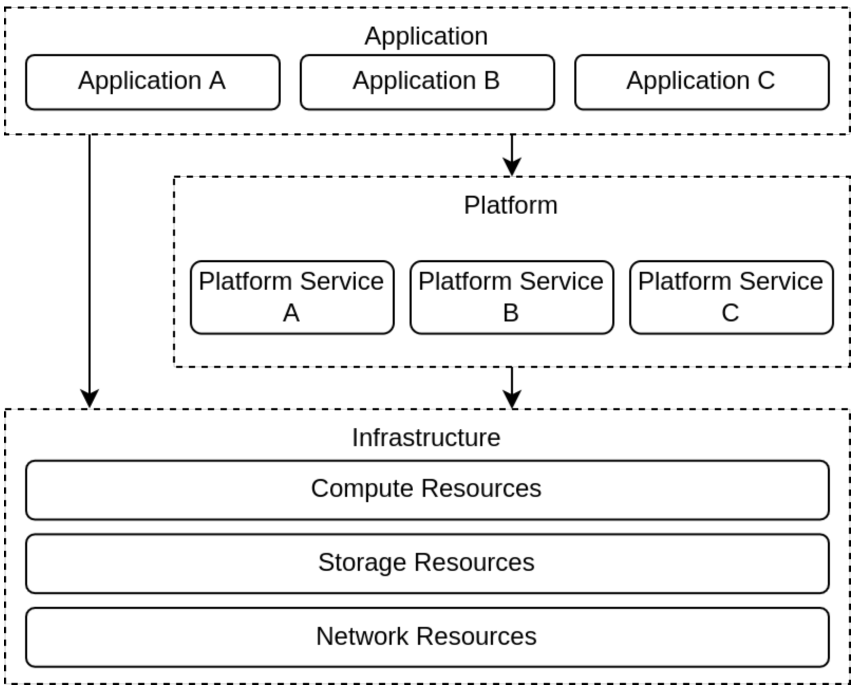

Figure 3.2: Layered model of a distributed computing system.

This section defines a generalized distributed computing model architecture, consisting 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 layer 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 abstracts 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, the platform layer in red, and the application layer in green.

Infrastructure Layer

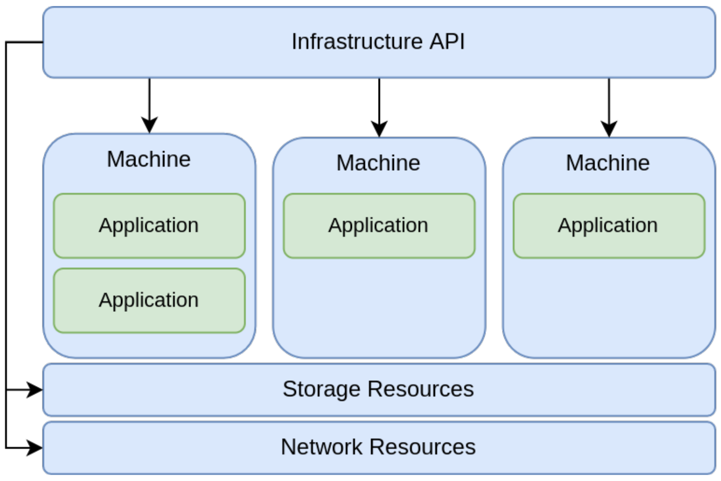

Figure 3.3: Overview of the infrastructure layer in the traditional distributed computing model.

While various kinds of user interfaces can be provided to tenants, such as websites, web portals, terminal user interfaces, graphical user interfaces, software development kits, and libraries, these user interfaces generally facilitate an application programming interface (API) that exposes resource management actions. To keep this model general, this model assumes the presence of an API.

Typically, computing resources provided by the infrastructure layer are in the form of machines, which combine CPUs, memory, peripheral devices, and ephemeral (machine-local) storage. 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. While compute resources provide execution environments for applications, storage resources allow storing data independent of a machine's lifecycle, and network resources enable connectivity between infrastructure resources. To keep this model abstract, we will not make any assumptions about the specifics of storage and network resources.

However, infrastructure resources can be provided in two forms: physical or virtual. Protection of data in physical infrastructure resources can be implemented for example by securing the physical location of the hardware, and facilitating application transparent encryption. Virtual infrastructure is protected by the isolation that virtualization systems, such as hypervisors, 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. 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 [23, 25].

Because the service provider manages the physical hardware and virtualization systems, the protection of infrastructure layer resources 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 monitor, allocate, and release resources to scale dynamically depending on the current needs.

Authentication and authorization
: Provide 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
: Provide means for storing and accessing data by managing storage resources of the infrastructure layer. Often includes caching, replication, and synchronization services to maximize availability, integrity, and performance.

Service discovery and load balancing
: Expose applications as network services to other applications and distributing traffic to multiple instances 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) manages 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 to automate the management and simplify access to those resources. They often also provide more complex behaviors like data management, application-level communication, service discovery, and load balancing. These services need the privilege to allocate, release, and monitor infrastructure resources.

Security
: Services that offer application-level authentication, authorization, and encryption to ease the process of implementing security into an application. These services manage the identities of applications and users, control access to resources and services, and generate and manage cryptographic keys.

Monitoring and Logging
: Monitoring and logging services collect monitoring metrics and logs from applications.

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

The usage of platform layer services provided by a service provider is optional. Application owners 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 to automate specific processes, be more cost-efficient, and reduce the complexity of the application layer.

Application Layer

The final layer is the application layer, consisting of the application owners' applications. These applications can also be in the form of services typically located in the platform layer and managed by the service provider. 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 application owner manages the applications of the application layer, while the service provider operates the platform layer services.

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, the ecosystem surrounding Kubernetes has been rapidly growing. It provides general platform services that fall in the infrastructure management and application orchestration categories, 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 a separate infrastructure layer, such as a Cloud IaaS offering.

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 services.

In this example, the Cloud provider takes on the service provider role.

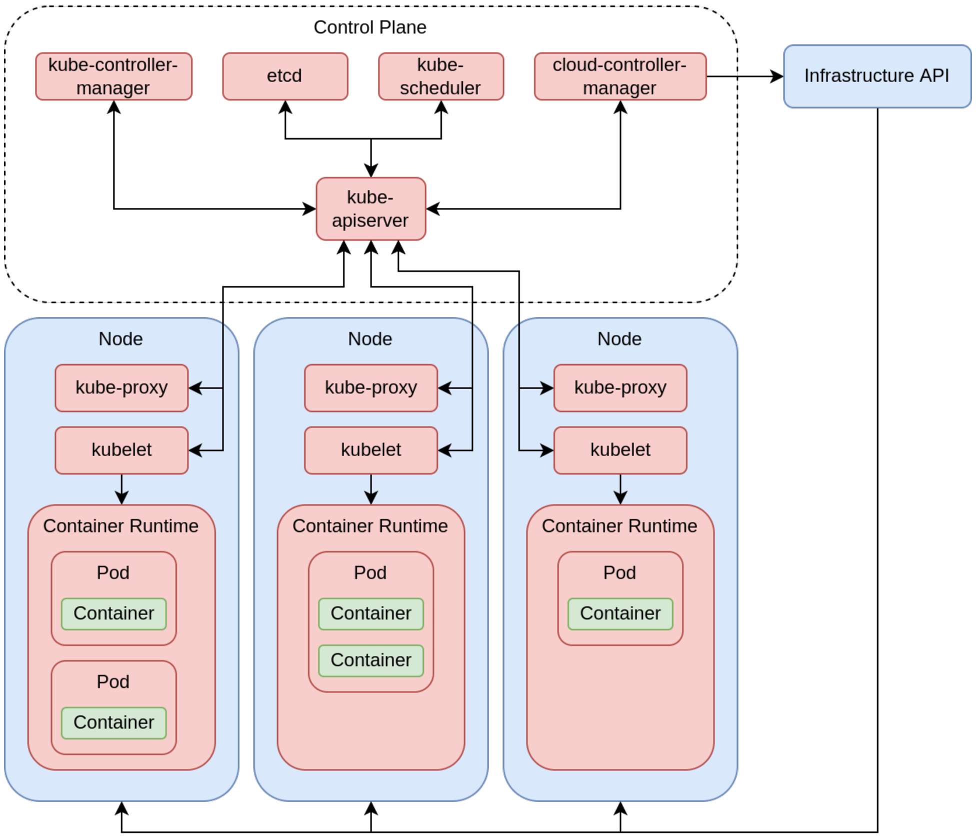

Figure 3.4: Overview of core Kubernetes components.

We will first look at an overview of 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, pods are logical execution environments in which applications, in the form of containers, can be deployed and executed. As such, pods are managed execution environments of the platform layer that contain application containers of the application layer.

Nodes

Nodes are machines running containerized applications. The following components

[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`

are present on each node: kubelet, container runtime, and 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 responsible for running containers. Kube-proxy is a network proxy that implements the discovery of applications inside the cluster and the exposure of those applications as network services.

Nodes correspond to compute resources of the infrastructure layer. However, the components on those nods implement application orchestration and infrastructure management capabilities and are part 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 for all cluster data (etcd), 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 for changes when needed. While each controller is a logically separate process, 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 control plane stores the desired global state of the cluster and requests modifications of the cluster to reach the desired state upon request of a tenant or change of the actual cluster state. These modification requests are either processed by another control plane component, such as the cloud-controller-manager allocating a new node by calling the infrastructure layer API, or components on existing nodes, such as the kubelet creating a new pod.

Infrastructure Management

Kubernetes enables Cloud providers to implement Cloud specific infrastructure management services by developing a cloud-controller-manager, Container Storage Interface (CSI) plugins, and Container Network Interface (CNI) plugins, allowing tenants to automatically scale their Kubernetes cluster depending on the current need.

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

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 in conjunction with their storage resources. These encryption services use keys provided by a KMS. While some Cloud providers allow tenants to utilize their self-managed KMS, the encryption service still needs plain text access to keys 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 CNI plugins also allow tenants to enforce network policies in the cluster, and transparently encrypt data traversing the network resources in order to protect confidential data.

Application Orchestration

The basic workflow of how Kubernetes orchestrates application containers on nodes is as follows. Note that this description of the workflow is greatly simplified.

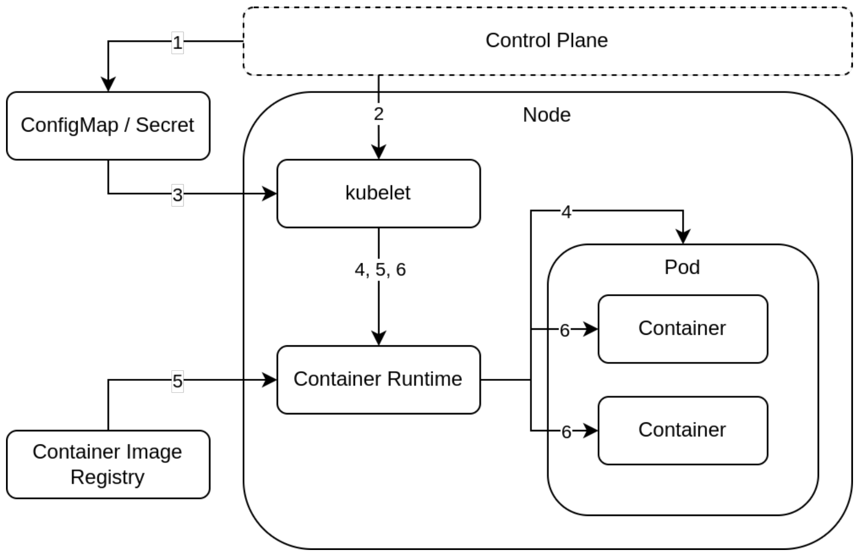

Figure 3.5: Kubernetes' application deployment, configuration, and execution workflow.

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 specification 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 ConfigMaps/Secrets.
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

Moving on from the traditional distributed computing model, this section gives an overview of the current state of confidential computing.

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

Confidential computing provides hardware-based primitives that allow the creation of trusted execution environments (TEEs). These TEEs have specific properties that protect data in use.

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 *Confidential Computing Consortium* [9] are:

Data confidentiality
: Prevent unauthorized entities from viewing data that is in use within a TEE.

Data integrity
: Prevent unauthorized entities from adding, removing, or changing data while it is in use within a TEE.

Code integrity
: Prevent unauthorized entities from adding, removing, or changing code executed in the TEE.

Besides ensuring the confidentiality of data, code integrity is also critical for the confidentiality of data. Even if data confidentiality is implemented correctly, executing compromised code inside TEEs can also lead to confidential data leakage.

Provided that the application code implements the computation correctly, data and code integrity ensure that neither the data nor the application has been modified, allowing clients to trust the results of the computations run inside TEEs.

As TEE technologies widely differ in their implementations, this thesis will treat the hardware and software components that create and protect TEEs as a single platform consisting of all hardware and software components involved, and refer to it as the “TEE platform”.

Besides the three core properties, TEE platforms also often provide mechanisms that enable the integration of a remote attestation process (see Section 3.3), enabling clients to assess the trustworthiness of TEEs created by an untrusted party. These technologies generally produce evidence about the authenticity and integrity of both the TEE platform and specific TEEs.

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 almost all software components between the hardware and the TEE from the list of trusted components, including system software such as the operating systems and hypervisors. The only components that need to be trusted are the components of the TEE platform.

Today most TEE implementations still rely on firmware components that are part of the TEE platform, allows manufacturers to deploy bug fixes and security patches. Because an untrusted service provider can compromise these firmware components, TEE platforms generally facilitate hardware-based mechanisms that produce evidence about the integrity of the TEE platform.

Memory Protection

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

Memory Encryption
TEE technologies rely on hardware components 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 [22], TEE technologies transparently en-/decrypt data. Most importantly, the unencrypted data is only available while being processed by the CPU, and is encrypted before leaving the CPU. 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 physical access to the machine only see encrypted data.

Memory Access Control
On the other hand, memory integrity is guaranteed by enforcing that only the TEE owning a specific memory region can modify data stored in this region. In order to achieve this, the TEE platform has to keep track of protected memory pages, and their owners.

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 confidential 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, including:

- 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. SDKs and frameworks often hide the split between host and enclave from developers to ease the development of such applications [27].

Library OSes like Gramine and Occlum go even further and provide a POSIX-like runtime environment with network, file I/O, and multithreading support. Because applications running inside the enclave do not have access to the underlying OS, library OSes provide libraries that implement OS system calls in the form of library functions, while a boilerplate host provides I/O functionalities [31].

Even though these SDKs, frameworks, and library OSes ease the development, using process-based TEEs still requires more development effort, and porting existing applications often still requires the modification of the application.

VM-based TEEs

The central concept of VM-based TEEs is to apply the TEE properties to a whole virtual machine. Traditionally, hypervisors are fully responsible for managing VM memory and thus have access to VM memory. On the other hand, TEE platforms offering VM-based TEEs take away the responsibility of VM memory protection from the hypervisor because the TEE platform itself already implements memory protection. So instead of having full access to a VM's memory, the hypervisor now only manages VM memory through mechanisms offered by the TEE platform.

VM-based TEEs are specifically designed to protect VMs from the rest of the system, including:

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

Comparison

Figure 3.6 compares the list of trusted components of an application running without confidential computing, inside a process-based TEE, and inside a VM-based TEE. In both TEE models the TEE platform enforces the TEE properties and thus the hardware has to be partly trusted.

On the one hand, process-based TEEs allow fine-grained separation of trust by splitting applications into host and enclave, but this requires applications to be ported to a new programming model. On the other hand, VM-based TEEs have much larger attack surfaces because they include an entire OS but allow complex applications to be run in a more secure environment without the need to modify the applications.

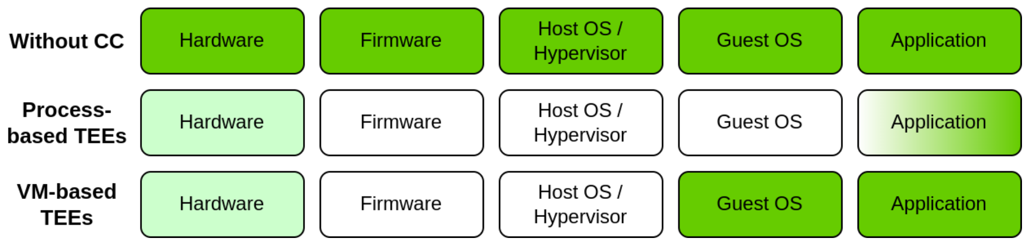

Figure 3.6: Comparison of TEE models. In both TEE, cases the hardware has to be partly trusted, illustrated by the lightly marked hardware components. In the process-based TEE model, the application has to be split into enclave and host, which is why the application component is marked with a gradient.

3.2.3 Commercially Available TEE Technologies

Intel SGX

Intel Software Guard Extensions (SGX) provides process-based TEEs by relying on hardware to create enclaves that contain application code and confidential data. As the name implies, it is an extension of Intel's CPU instruction set architecture [10].

The CPU protects a designated memory area called the Processor Reserved Memory (PRM) established by SGX by ensuring that other software and hardware components, such as system software (hypervisor or OS) and DMA devices, do not have access to the PRM. Confidential data and enclave application code is stored in the Enclave Page Cache (EPC), a subset of the PRM. SGX relies on 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 an 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 to prevent leaking confidential data through PME attacks and guarantee the confidentiality of data stored in EPC pages after eviction.

Initially, the system software asks 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 firmware components and the EPC pages of the enclave, producing attestation evidence which is then signed by the CPU using a cryptographic key that is only accessible by the CPU. The signature can then be used to verify the authenticity of the evidence, which in turn can be used to verify the integrity of the enclave and the SGX platform. An attestation can not only be requested after initialization but also during runtime.

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 are tagged with an Address Space Identifier (ASID). The CPU uses the ASID to restrict data access and modification to the owner with the same ASID, protecting data from any unauthorized usage. However, in the first iteration of SEV, the registers of a virtual CPU could be used to leak confidential data when shutting down a VM. Subsequently, AMD released their second iteration SEV-ES (Encrypted State), which encrypts VM memory and virtual CPU registers. The latest version SEV-SNP (Secure Nested Paging), introduced additional features that protect VM memory integrity.

The attestation process for SEV VMs is similar to the attestation process for SGX enclaves. A hypervisor launches a VM, and after the VM is fully loaded, the VM's memory is encrypted. Subsequently, the AMD-SP measures SEV firmware components and VM memory pages, which are signed using a cryptographic key only accessible by the CPU. Again, the signature can then be used for verifying the authenticity of the measurements, and the measurements can be used for verifying the integrity of the VM and the SEV platform. 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 for measurements at any time, enabling a more flexible remote attestation.

3.2.4 Limitations

Performance Impact

Existing solutions require careful configuration to achieve acceptable performance or are inappropriate for specific use cases [1]. For example, due to the limited size of the EPC and the restricted programming model, Intel SGX displays a significant performance loss for high performance workloads, making the usage of SGX for these kinds of cases impractical.

CPU centric focus

Most of today's confidential computing solutions focus on a CPU-level view of memory permissions. This limits the application of confidential computing to heterogeneous computing systems, where discrete accelerators are used in order to speed up specific computations (e.g. GPUs and NPUs for machine learning workloads). However, there is ongoing work on integrating confidential computing into heterogeneous computing systems [15].

New technology that requires further research

Since the introduction of Intel SGX in 2015, numerous vulnerabilities have been found in the SGX architecture [12]. 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 operating systems, hypervisors, and VM firmware, to implement the support of these technologies. These implementations also require further research and testing to find vulnerabilities and security issues.

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 give an overview of the remote attestation process.

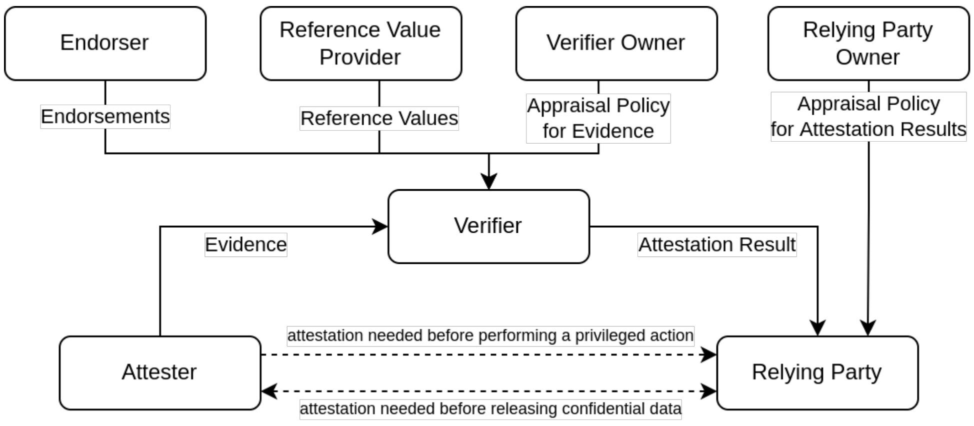
Endorser
Reference Value Provider
Verifier Owner
Relying Party Owner
Endorsements
Reference Values
Appraisal Policy for Evidence
Appraisal Policy for Attestation Results
Verifier
Evidence
Attestation Result
attestation needed before performing a privileged action
Attester
Relying Party
attestation needed before releasing confidential data

Figure 3.7: Remote attestation roles and data flow.

Remote attestation begins with the relying party requesting a verification of the attester. Subsequently, the attester generates evidence about its trustworthiness, which the verifier uses in combination with endorsements from endorsers and reference values from reference value providers by applying an appraisal policy to assess the trustworthiness of the attester. After receiving the attestation results from the verifier, the relying party 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. *Remote ATtestation procedureS (RATS) Architecture* outlines how the trust between those two roles can be established [8, Section 7.1]

3.3.1 Chain of Trust

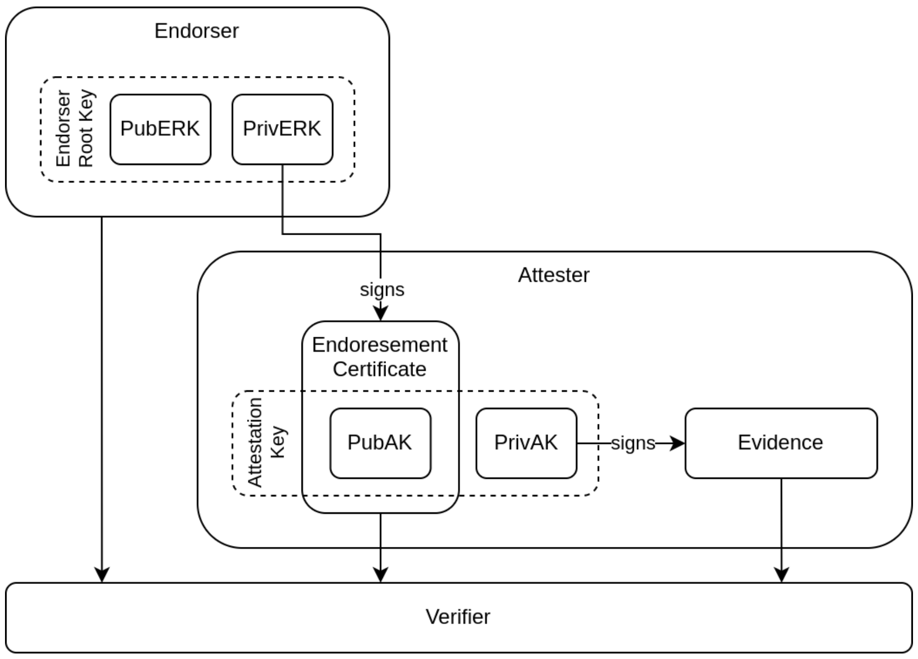

Figure 3.8: Remote attestation chain of trust.

Typically, a verifier comes to trust an attester indirectly by having an endorser, creating a chain of trust. The root of the chain of trust is 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 must be stored in tamper-proof hardware 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 the 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 attester produced the evidence 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 signature.

Finally, after validating the authenticity and integrity of the evidence, the verifier appraises the evidence and conveys the attestation results to the relying party.

3.3.2 Secure Communication Channel between Attester and Relying Party

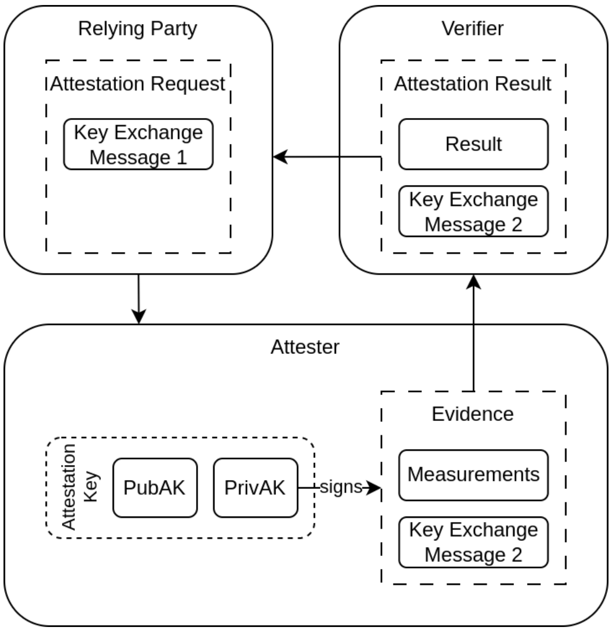

Figure 3.9: Integration of the Diffie-Hellman key exchange protocol into the remote attestation process.

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). To simplify the illustration, we will assume that the prime number p and generator g are already specified by the protocol or have already been 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 pub_A. The attester then includes the second key exchange message pub_B in the produced evidence. Because the evidence is signed using the attestation key, the key exchange message is part of the chain of trust, allowing the verifier to validate the authenticity and integrity of the key exchange message. After successful verification, the verifier includes the key exchange message in the attestation result. Now pub_A and pub_B have been exchanged between the attester and the relying party, allowing the derivation of the shared secret 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 failing.

While this illustration is about the establishment of a secure communication channel in the remote attestation process, current ongoing work by Tschofenig et al. [32] 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 clients, and defined a generalized layered architecture. This chapter first describes a threat model, identifying threats and issues of the infrastructure and platform layers when the service provider becomes untrusted. The threat model is followed by changes that have to be made to the traditional trust model and architecture, integrating confidential computing and remote attestation to address the threats and issues.

Section 4.4 summarizes the changes into distinct requirements of the trusted distributed computing model, and Section 4.5 evaluates the model using the threat model previously defined.

Finally, this chapter looks at two projects that are working on implementing the trusted distributed computing model into the Kubernetes architecture, previously established in Section 3.1.5.

4.1 Threat Model

The now untrusted service providers administer the infrastructure and platform layer defined in Section 3.1.4. In this threat model, we will identify threats originating from the resources and services of these two layers.

4.1.1 Infrastructure Layer

While looking at the infrastructure layer, we will also discuss why traditional mitigations fail in protecting data from untrusted service providers.

Threats from storage resources

Traditionally, the service provider protects storage resources by securing the location of the physical hardware and storage systems that provide application-transparent data encryption. These measures prevent external attackers, but not the service provider, from gaining physical and plain-text access to confidential data. Therefore, data encryption has to be implemented at application level.

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 sensitive information using untrusted communication channels need to implement authentication, authorization, and encryption to use these untrusted communication channels and not leak confidential data to a malicious attacker that has access to the underlying network resources. Authentication and encryption are usually implemented using the Transport Level Security (TLS) protocol [11], which utilizes a trusted third party (the certificate authority) for authentication, and the Diffie-Hellman key exchange protocol to establish a shared secret for encryption.

Typically, these implementations do not rely on infrastructure layer resources and are implemented in the application layer.

Threats from compute resources

Techniques for protecting data resting in storage and traversing networking resources are primarily based on cryptographic primitives, such as encryption and digital signatures. These primitives require a cryptographic key (e.g., secret key for symmetrical encryption, private keys for asymmetrical encryption and singing) that needs to be kept secret. Typically, these keys and the decrypted data are stored in a machine's memory while in use, making memory a prime target for attackers.

There are two types of physical attacks related to a machine's memory, summarized by Weis [33]: **Direct Memory Access (DMA) Attacks** exploit the design of the x86 instruction set architecture that allows hardware subsystems to bypass the CPU and directly access the memory of the machine to improve performance. This might lead to maliciously designed DMA devices to access or modify memory regions that contain confidential data. As the name implies, **Physical Memory Extraction (PME) Attacks** directly extract data from the physical system memory of a machine. Examples for this type of attack are monitoring the memory bus of a machine, cold boot attacks, and exploiting 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.

One primary benefit that virtualization brings is isolation with the goal of shielding a VM from other VMs. Traditionally, the hypervisor is a trusted high-privileged software component responsible for the virtualization of virtual CPUs, mapping of a VM's virtualized physical memory address space to the host machine's physical memory address space, intercepting and carrying out privileged operations invoked by a VM, and providing VM management tasks such as starting, stopping, suspending, restoring, and migrating VMs. But a malicious administrator or an attacker who 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, many vulnerabilities have been found in commonly used hypervisors that break the isolation promises of hypervisors [23, 25]. We will refer to this kind of attacks as **Virtualization-based Attacks**.

Measures for protecting VMs from the hypervisor mostly separate the high-privileged operations into separate components that control the access to these operations and move the rest of the hypervisor into a less privileged execution mode [16, 30, 19, 21]. 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, and they are also unable to 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 application owner can often provide a machine's operating system, the service provider typically controls the firmware. BI attacks exploit the high level of privilege of firmware and the fact that firmware may be invisible to the operating system to compromise security measures in higher levels of software. As such, establishing trust in remote machines requires every piece of software executed during the machine's lifetime (including the firmware) to be verified or isolated.

There have been two main approaches to preventing BI attacks:

The **Secure Boot** approach requires each software component needed for the boot process, including firmware, bootloader, and kernel of the operating system, to be signed. During boot, the signature of these software components is then verified. When a software component has been modified, the verification fails, leading to the abortion of the boot process. This also requires the maintenance of a certificate which is used to verify the signatures. 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 store measurements of loaded software components. Various software components, such as the firmware, bootloader, or kernel, take these measurements. After a successful 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 (often the firmware) to verify or measure other software components required for the boot process. However, there is no such software component in the traditional distributed computing model, as the service provider can modify all software components.

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 threats of the infrastructure layer also apply here.

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

Monitoring and logging services aggregate metrics and logs of applications. While monitoring metrics usually do not contain sensitive information, application logs sometimes contain sensitive information that can lead to the leakage of confidential data.

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

A malicious service provider might misuse application orchestration services to leak confidential data by:

- executing malicious code in the provided target environments.
- modifying applications or their configurations, influencing the behavior of those applications.

Service Type	Threats
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: Overview of platform layer threats.

4.2 Trust Model

The following section describe the changes made to the trust model of the traditional distributed computing model.

The roles of the trusted distributed computing model are still present, with the difference that the service provider is no longer considered trusted. However, there are now three roles from the RATS framework (see Section 3.3) present:

Hardware Manufacturer
The hardware manufacturer provides hardware to the service provider that forms the base of the infrastructure layer. The hardware has to be able to create TEEs, and the hardware manufacturer endorses the security of those TEEs. The hardware manufacturer takes on the endorser role defined by RATS. 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 provide infrastructure layer resources and manages platform layer services but now utilize a TEE platform provided by the hardware manufacturer to offer the capability of creating TEEs and getting evidence for the integrity of the TEEs.

Reference Value Provider
Reference value providers generate reference values for TEEs in advance, which are used by the verifier to validate the integrity of TEEs created 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.

Application Owner
The responsibilities of application owners stayed the same. However, because the service provider is untrusted in this model, 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 the services of the platform layer.

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 owner. When a relying party, usually the application or data owner, requests the verification of a TEE, the verifier validates that the 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 the control of the service provider. So in order for the verifier to fully validate the integrity of the TEE, the TEE platform's integrity also needs to be verified.

The verifier is only responsible for verifying the integrity of TEEs as a whole and does not verify individual software and applications inside TEEs.

4.3.2 Infrastructure Layer

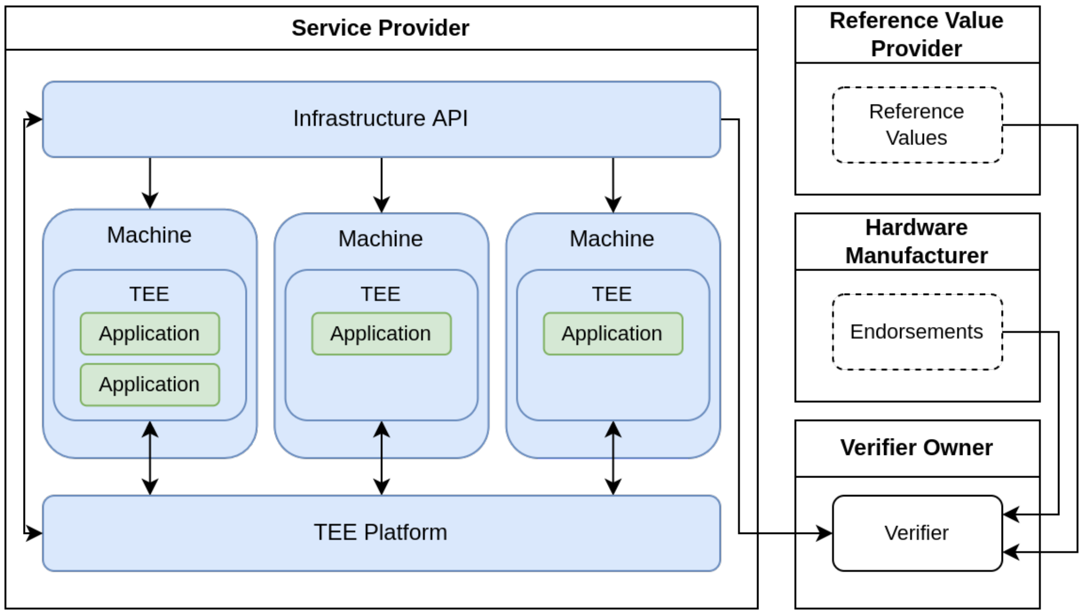

Figure 4.1: Overview of the infrastructure layer in the trusted distributed computing model.

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

Instead of directly running applications on provided (physical or virtual) machines, data and application owner rely on TEEs to protect data currently used by applications. In Section 3.2, we have seen two TEE models. On the 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 of producing evidence of software components that have direct access or control the access to the memory of a TEE, such as the VM firmware and firmware components of the TEE platform. The hardware manufacturer endorses the capabilities of securely creating TEEs and producing evidence.

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.

Because in this model, the service provider is untrusted, there are two possible solutions to the threats of the services traditionally located in the platform layer:

- Moving these services to the application layer or implementing these services directly into applications.
- Splitting the service into privileged and unprivileged parts. The unprivileged parts can still be operated by the service provider, while the privileged parts have to be managed by the application owner.

The first option can be applied to all services of the platform layer. Therefore, this section will focus on evaluating each previously defined service category using on the latter option.

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 service provider, as application-level security depends on those services. Management of identities, administration of authentication policies, and data encryption and decryption are 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 controlled by the untrusted service provider, the software responsible for managing the lifecycle of TEEs is also untrusted (e.g. hypervisor, OS managing process-based TEEs). An attacker can also modify applications and application configurations 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 how to deploy applications into TEEs:

The first option is to create an application package that the TEE platform can directly deploy. For example, the application owner might package an application, including its (non-confidential) configuration, straight into a VM image that the service provider can deploy as a VM-based TEE. The application owner has to provide reference values in the form of measurements of the VM to 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 configurations.

The second option is to 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 tenants can deploy 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 independent 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 discussed how application-level authentication, authorization, and encryption of confidential data are 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.

As mentioned before, application logs must also be protected if they include sensitive information. This can be achieved by encrypting application logs or obfuscating sensitive information.

4.4 Requirements

This section summarizes the changes discussed in the previous two sections by defining 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. This has to include evidence for the integrity of:

- R4.1 individual TEEs.
- R4.2 the TEE platform.

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

R5 the integrity of individual TEEs using evidence provided by the service provider and reference values from the reference value provider. This includes validating the authenticity of the evidence using the endorsement from the hardware manufacturer.

R6 the integrity of the TEE platform using evidence provided by the service provider and reference values from the hardware manufacturer. This includes validating the authenticity of the evidence using the endorsement from the hardware manufacturer.

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 securely 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 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. Implementing authentication, authorization, and encryption in the application layer prevents attackers that have access to storage and network resources from gaining 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 performs measurements and verification. R3 and R4 require the service provider to offer the capability to create TEEs and get evidence about its integrity by facilitating a TEE platform. The TEE platform is provided and endorsed by the hardware manufacturer (R1 and R2). The TEE platform takes on the role of the trusted component performing measurements, while verification is performed by the verifier.

Physical access to a machine's memory is prevented as data leaving the CPU is encrypted before storing it in memory. As such, attackers with access to the physical memory of a machine still does not have plain-text access to the data, preventing DMA and PME attacks.

The TEE platform also mitigates virtualization-based attacks. VM-based TEEs split off the hypervisor's memory access control and isolation tasks into the TEE platform. The hypervisor is still provided with interfaces to manage VMs (e.g., starting, stopping, suspending), but the TEE platform implements the isolation guarantee. 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 [28]. 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 is relayed to the verifier in order to verify OVMF's integrity. The attestation report also includes evidence on the integrity of SEV's firmware components, fulfilling requirement R4 and 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., Intel SGX), in which case the VM itself and the hypervisor are not considered trusted. Again, isolation is provided by the TEE platform 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, mitigating virtualization-based, and BI attacks.

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.

Application orchestration still relies on untrusted software to manage target execution environments for applications. In this model the target environments are VM-based TEEs managed by a hypervisor or process-based TEEs managed by the surrounding host (or guest) OS. R9 requires that the application owner controls the rest of the application orchestration process, that is, installation, configuration, and execution of applications inside provided TEEs. It also requires the verification of both TEEs and applications before supplying applications with confidential data. On the one hand, verifying applications allows the application owner to detect modifications of applications and their configurations. On the other hand, removing the service provider from the process of deploying applications into TEEs, prevents the service provider to execute malicious code in TEEs.

While these requirements remove the service provider from the list of trusted entities, the requirements also 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/`

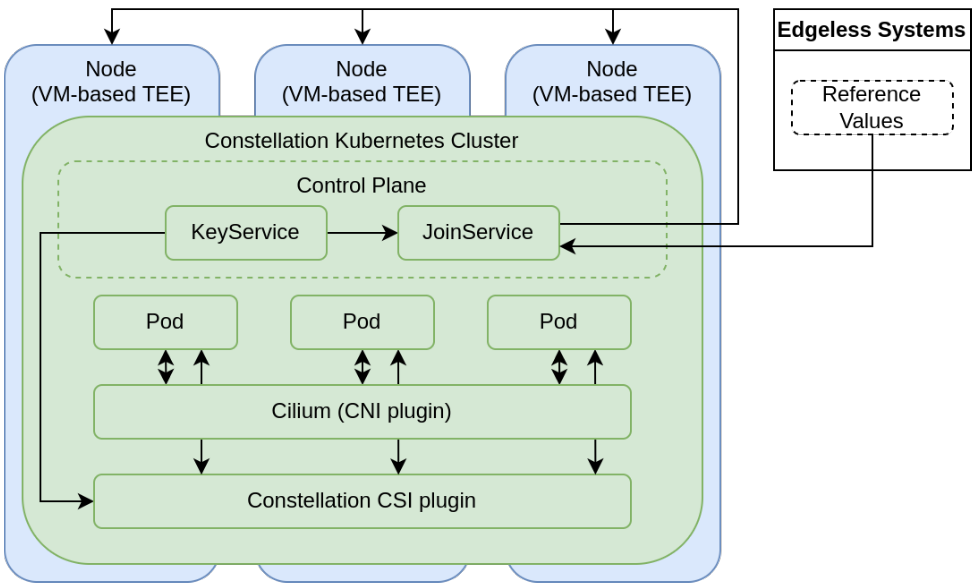

Figure 4.2: Constellation Kubernetes overview.

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 which are managed by the KeyService. These keys are then used to encrypt etcd data 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 example, AWS currently does not provide the capability of providing SEV-based VMs, not meeting R3. While Azure and GCP offer SEV-based VMs, Azure currently does not facilitate reviewable VM firmware, not meeting R4.2, and GCP does not provide evidence produced by the SEV platform, not fulfilling R4.

Because the Cloud providers do not meet R3 and R4, the JoinService (verifier) can not meet 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 satisfy R4, the verifier can not fully verify the integrity of the TEE platform or individual TEEs, preventing the verifier from satisfying R5 and/or 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. Constellation meets requirements R7 and R9, because security, infrastructure management, and application orchestration services are all part of the Kubernetes cluster, 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 main 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, and the 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 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.

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

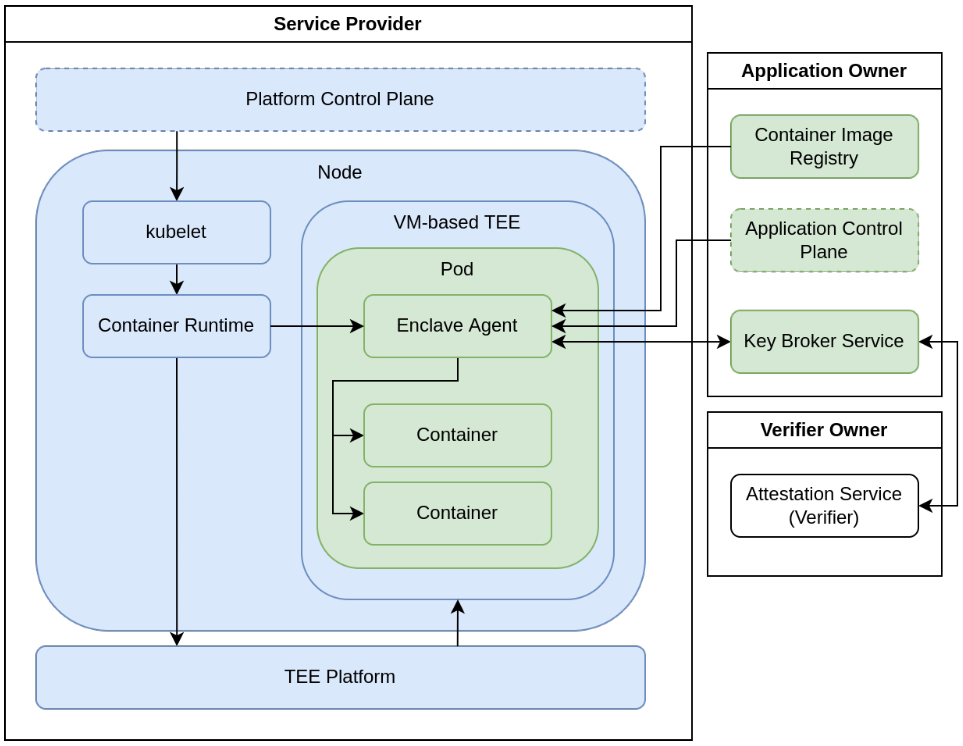

Figure 4.3: Confidential Containers application orchestration overview.

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 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 enclave agent enforces the split of the control plane by blocking privileged actions originating from the platform control plane, such as creating containers 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 to securely release 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 secure communication with other components, such as the application control plane.

It is still unclear, 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 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 is successful, the key broker service releases keys and a pod specification policy to the enclave agent. The policy can include, for example, storage and network configurations, a container image whitelist, and a certificate that 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 does the enclave agent create the containers inside the VM and provide 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.

While the overall architecture is there, Confidential Containers is still far from production ready. Specific implementations are still missing or under discussion, for example, the components of the remote attestation process, mainly the key broker service and the enclave agent, are still under heavy development, and implementation details of the dual control plane architecture and the verification of pod specifications are still under discussion. These issues require changes in the whole Kubernetes architecture, from the API down to the container runtime. Before these issues have been resolved, Confidential Containers does not fully meet requirement R9.

5 Conclusion

The aim of this thesis was the introduction of a new trusted distributed computing model that protects the confidentiality of tenant data from untrusted service providers. Using a threat model that identified threats and issues that arise when service providers become untrusted, this thesis evaluated the trusted distributed computing model, which integrates confidential computing and remote attestation into the traditional distributed computing model. The evaluation showed that confidential computing combined with remote attestation could solve the threats and issues of the traditional model.

However, the removal of the service provider as a trusted entity comes with a trade-off. While 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 compromise the confidentiality of data back to the application owner.

Additionally, current confidential computing technologies still have performance limitations that either lessen the cost-efficiency aspect of facilitating distributed computing systems or make using confidential computing technology impractical.

Furthermore, confidential computing technologies are still relatively new. Implementations of TEE platforms, remote attestation protocols, and their support in other software components are still in the early stages of development. Further research might uncover new vulnerabilities that require changes in implementations and design choices. Therefore, protocols and implementations are all still subject to substantial changes. 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 the infrastructure and platform layers are 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 that meet the increasing demand for more privacy.

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] 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/.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] ARM Limited. *Arm Confidential Compute Architecture Software Stack Guide*. Version r1p0. Sept. 13, 2022.

[21] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

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

[28] Andras Slemmer and Stefan Deml. *Swiss cheese to cheddar: securing AMD SEV-SNP early boot*. July 2022. URL: https://blog.decentriq.com/swiss-cheese-to-cheddar-securing-amd-sev-snp-early-boot-2/.

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

[30] 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.

[31] 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.

[32] 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/.

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

[34] 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.

[35] 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.

[36] 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.