# Using Recursive Attestation to Scale Trust in Modern Heterogeneous Cloud Architectures

Yaoxin Jing
y.jing24@imperial.ac.uk
Imperial College London

Michael Steiner
michael.steiner@intel.com
Intel Labs

Anjo Vahldiek-Oberwagner
anjovahldiek@gmail.com
Intel Labs

Mona Vij
mona.vij@intel.com
Intel Labs

Lluis Vilanova
vilanova@imperial.ac.uk
Imperial College London

## Abstract

Modern cloud infrastructures are increasingly complex, driven by heterogeneity, disaggregation, and dynamic service composition—exposing critical limits in traditional attestation models. These models struggle to scale when trust must span multiple domains and elastic services. We present scale-out attestation, a paradigm decoupling platform trust verification from app-level attestation. Our design introduces a recursive attestation framework leveraging abstract service identities and trusted deployment workflows: a single infrastructure agent verifies platforms via abstract policies, while services derive instance-agnostic identities enabling secure recursive dependency attestation. We implement the system on FractOS, a distributed OS for disaggregated data centers, and plan to extend Confidential Containers for practical deployment. Evaluation shows strong security with minimal overhead, enabling scalable confidential computing across heterogeneous and dynamic cloud environments.

## Keywords

Remote Attestation, Confidential Computing, Trusted Heterogeneous Disaggregated Could

**ACM Reference Format:**
Yaoxin Jing, Michael Steiner, Anjo Vahldiek-Oberwagner, Mona Vij, and Lluis Vilanova. 2025. Using Recursive Attestation to Scale Trust in Modern Heterogeneous Cloud Architectures. In *16th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '25), October 12–13, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3725783.3764390


*APSys '25, Seoul, Republic of Korea*

ACM ISBN 979-8-4007-1572-3/25/10
https://doi.org/10.1145/3725783.3764390

## 1 Introduction

Confidential computing (CC) has become a fundamental tool to support important classes of privacy-sensitive apps in the public cloud, such as finance and healthcare [2]. At the core of CC, apps use remote attestation to ensure the integrity of all relevant parts of the hardware/software platform, establishing trust to perform the intended computation [17, 46].

Existing remote attestation solutions assume *ahead-of-time knowledge of all trust relationships* [3, 6, 7, 10, 12, 25, 29, 40–42, 52, 55], making them inadequate to address the heterogeneity and modularity of modern systems. Consider a confidential cancer image recognition app with a CPU-based frontend passing CT scans to a GPU-powered inference service from Tempus AI [1], using external Azure Storage [8]. This highlights two key challenges: (i) platform heterogeneity, and (ii) composition and scheduling decisions, causing a *combinatorial explosion* of trust measurements across all software and hardware stack combinations.

*Platform heterogeneity* affects measurements across all components: (i) different hardware vendors and attestation primitives like Intel SGX [27], AMD SEV [4], or Arm CCA [5]; (ii) specialized devices like GPUs and TPUs with device-specific attestation [38, 39]; and (iii) different platform software versions for firmware and OS [54]. For instance, inference engines may use NVIDIA MIG attestation [38] within Linux CVMs with ARM CCA [5], while Azure Storage runs on Intel Gramine-TDX [26, 31], creating fragmented trust mechanisms difficult to verify with traditional attestation.

*Composition and scheduling decisions* affect measurements across all components too. The pervasive use of service-oriented architectures (SoA) provide scalability, elasticity, and modularity [24, 32, 36], but this level of service composition also requires our app to provide measurements for every possible instance of the third-party inference and storage services; this is impractical because dynamic instances of an elastic service cannot be known

in advance, and is unnecessary because a single app will often be directed by a request scheduler to a small subset of the service instances. In addition, VM memory configurations are injected as BIOS changes, which affect system boot measurements [21]. Furthermore, service composition can also happen in a recursive way that is opaque to clients [35]; imagine, for example, that the storage service provides filesystem semantics, and internally uses a third-party HiPPA-compliant block storage service, similar to Azure Files [9]. In this case, our app can only verify the correctness of the file system if it knows and attests the block storage service, even though it cannot directly access it. Finally, the promised cost reductions of hardware disaggregation [20, 44] magnify the combined effects of platform heterogeneity and resource scheduling, since devices are dynamically assigned to apps from a large, heterogeneous pool.

Other attestation solutions, like Marblerun [45], avoid ahead-of-time knowledge of all trust relationships by placing the entire operator management stack inside the app's TCB. However, this enlarges the TCB and hinders operators from evolving their their platform management tools independently of tenant apps.

In this paper, we present a system that achieves scale-out attestation; the system avoids ahead-of-time knowledge of all trust relationships (and therefore avoids combinatorial explosion) by introducing new mechanisms for *recursive abstract attestation*. We make attestation a recursive process where: (i) services and apps (such as the example frontend) attest other services they use (such as the example inference and storage services), as well as attest the secure communication channels they use with each other; (ii) tenants provide encrypted deployments, which are managed by an untrusted scheduler and decrypted by a deployment agent (DA) once inside an attested execution environment; and (iii) the software/hardware platform of each deployment, including the DA, are attested by a trusted infrastructure agent (IA), which is in turn attested by each tenant (e.g., to ensure it only supports valid platform configurations, as in the configuration attestation service of Scone [22]). This recursive approach decouples tenant deployments from ahead-of-time knowledge of the platform, conscripting all checks of hardware-specific measurements to the IA.

We pair recursive attestation with *abstract service identities*, which together enable us to decouple service attestation from per-instance measurements. An abstract service identity is a unique attestable measurement that is shared across instances of the same service. All instances within the same abstract service identity are indistinguishable from an attestation client perspective (e.g., they have the same version and service instance configuration settings); as a result, the untrusted cloud operator can deploy scheduler, auto-scaling, endpoint routing, and load-balancing components without impacting attestation (such as the example inference and storage services with auto-scaling and load-balancing).

We argue that growing platform heterogeneity and dynamic service composition in the clouds make scale-out attestation infeasible ((§ 2)). To address this, we propose recursive abstract attestation design, prototyped on FractOS [50] (§§ 3 and 4), and conclude the paper with security and performance evaluation (§ 5).

## 2 Scale-Out Attestation Challenges

**Platform Heterogeneity.** Heterogeneous cloud apps are deployed across diverse compute platforms, such as CPUs, GPUs, FPGAs, TPUs, or DPUs. Each has distinct hardware attestation primitives and infrastructure software [4, 26, 39, 51, 56], which leads to a combinatorial explosion of the required reference measurements that current attestation schemes need [54]. Tenants must track every valid infrastructure software combination, making trust verification fragile, error-prone, and difficult to scale. Moreover, attestation requires tenants to verify vendor-specific hardware attestation primitives, forcing tenants to handle complexities they should not deal with [3, 7, 12, 21, 52].

**Opaque Service Composition.** In service-oriented deployments, apps are built from multiple third-party services that themselves may recursively depend on additional internal services. These compositions are intentionally opaque for service clients because of security and operational reasons [18, 28, 33, 35, 37]. Current attestation models either require full access to all interacting services, such as Scone [6, 43], or only support composing a single level of trust, such as Marblerun [45]. As a result, they cannot establish trust transitively across services that are composed in a way that is either opaque (e.g., the file system and HiPPA-compliant block storage services in our example) or dynamic (e.g., when deploying an elastic inference service).

**Elastic Scheduling Decisions.** Modern cloud services rely on both static and dynamic scheduling to manage service instances (SI). In dynamic scheduling, SI are launched and terminated automatically on demand. These decisions introduce complexity in attestation because every new instance must be attested separately under traditional models [3, 6, 7, 10, 12, 25, 40–42, 52, 55]. This incurs latency and operational burden in large-scale systems. However, static scheduling, where specific instances are pre-assigned or manually configured, doesn't solve the problem. Even in such cases, slight differences in

deployment (e.g., due to hardware configurations, env. variables) may lead to inconsistent measurements [54]. This causes hash mismatches during attestation, breaking trust even though the logical service remains unchanged. Thus, scalable attestation must decouple trust from specific instances and tolerate configuration variations, motivating our proposed model.

## 3 Design

This section presents the design of our system, which is built on top of FractOS [50]. FractOS deploys small distributed OS instances called *controllers*, which run on their own TEE and mediate secure communication between apps and services, both within and across nodes. Note that we use the terms "app" and "service" interchangeably, as they have no distinction in our system. Each app executes within a TEE, can only directly communicate with their assigned controller via the network, and indirectly communicate with other apps through the communication primitives provided by controllers. The rest of this section first outlines the threat model (§ 3.1) and system overview (§ 3.2), followed by our distributed workflow for efficient scale-out attestation (§ 3.3–§ 3.5).

### 3.1 Threat Model

We identify three subjects on our system, which are shown in Fig. 1: the *tenant(s)*, the *cloud provider (CP)*, and the *FractOS operator (FO)*. A *tenant* owns one or more apps (shown in green) and does not trust any other party in the system; importantly, a tenant can distrust other tenants, such that the app owner can distrust the inference service owner. The *CP* is responsible for provisioning the physical infrastructure (gray), including TEEs, firmware/BIOS, and hardware devices, as well as untrusted infrastructure such as networking and resource scheduling (shown in red). The *FO* manages the FractOS infrastructure (orange), whose are described in the sections below; tenants do not trust the FO, but trust the components it deploys after attesting them. Each TEE is assumed to have a hardware root of trust, and can establish secure channels with other devices (e.g., GPUs) using protocols such as SPDM [23].

Adversaries in this model include malicious third-party apps owned by some tenant, app deployments compromised by a malicious *CP* (e.g., through the scheduler, host OS, or hardware), deployments compromised by a malicious *FO* (e.g., through a tampered controller), or network-based man-in-the-middle attacks. Attacks such as side-channel attacks, denial-of-service (DoS), and physical tampering are out of the scope of this work, but could be addressed using existing solutions [47, 57].

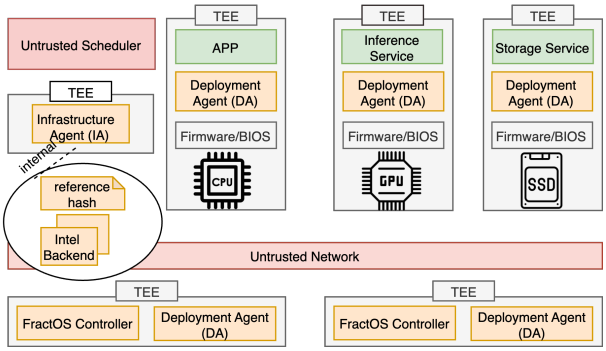


**Figure 1: Overview of the scale-out attestation system**

### 3.2 System Overview

Scale-out attestation aims to decouple platform attestation from service attestation, enabling scalable, recursive trust across resources in a disaggregated and heterogeneous cloud. This design supports abstract service composition while preserving the elasticity and modularity expected in modern cloud systems. To achieve these goals, the untrusted FO deploys three types of attestable components within TEEs (orange boxes in Fig. 1): the *infrastructure agent (IA)*, the *deployment agent (DA)* and the *FractOS controller*.

The IA and DA address *platform heterogeneity* by attesting components across the infrastructure. The IA has an *abstract attestation policy*, a series of platform-specific backends, a *cluster private key (CPK)* that is only released to attested DAs, and a *controller image verification policy (controller IVP)*, which describes a controller image with a public key for image verification, the image URL, and the image version. The abstract policy attests each DA, which is then delegated the CPK to deploy and attest additional components owned by the FO and other tenants (i.e., controllers and apps).

Both FO and tenants send deployment requests to the untrusted cloud scheduler. To ensure the integrity of the deployment, they attach an encrypted version of the deployment request (containing IVP, command line, and injected configuration files and environment variables) using the *cluster public key* (the counterpart to the CPK). The IA then releases the CPK to a successfully attested DA, which in turn extracts and decrypts the deployment request (which is therefore integrity-protected), verifies its contents (i.e., image integrity), and finally launches the requested program inside the TEE.

Controllers and app are managed slightly different by the DA. When *deploying a controller*, the FO does the following steps: (i) attest IA; (ii) upload controller IVP to the IA, which creates the secret CPK; and (iii) get the cluster public key to encrypt deployment requests. If the request identifies a controller (via an optional field), the

DA checks that the deployment request is valid according to the controller IVP, and then proceeds with the deployment. The DA will pass the CPK to the controller once deployed, which it will use to connect to other controllers using encrypted network transports [48, 53]. When *deploying an app*, the tenant does the following steps: (i) attest IA; and (ii) get the cluster public key to encrypt deployment requests. If the request identifies an app (via an optional field), the DA proceeds with the deployment. The tenant has the option of embedding a symmetric key on the encrypted request, which the DA will use to decrypt the app image. The tenant also has the option to control how apps can be co-located on the same TEE, by setting an additional field on the deployment request; the DA will check that all co-located apps have the same secret value on that field (the default is to disallow co-location). The DA will pass this optional key and secret fields to the app once deployed.

This separation of concerns abstracts the complexity of hardware platform attestation away from the FO and tenants, who only need to attest the IA, together with its abstract attestation policy and controller IVP. Since the CPK is only released to attested controllers, they do not need to attest each other, as well as tenants do not need to attest controllers. § 3.3 contains more details.

To support *elastic scheduling decisions and opaque service composition*, applications can check that any service instance they connect to, conforms to a reference *abstract service identity* that captures the relevant information that is constant across equivalent instances of the same service. An attestation report cannot be used in practice, since each instance might have different configurations that do not compromise the security of the service (e.g., memory size or thread count). Instead, the DA cryptographically hashes the app deployment request, which is privacy- and integrity-protected through the CPK; to support identity equivalence across instances, the DA only hashes a well-defined subset of the fields in the deployment request (see § 3.4 and § 3.5 later). The DA then sends this to the corresponding controller, which allows apps to check the identity of any instance against a reference value (calculated by tenants using the same method). Since identities are agnostic to the platforms and specific deployment parameters, they remain stable across dynamically instantiated or migrated services. Furthermore, each identity encodes a policy for how the service itself uses its security monitor to attest any other services it may use; this enables recursive attestation across arbitrarily deep elastic service compositions.

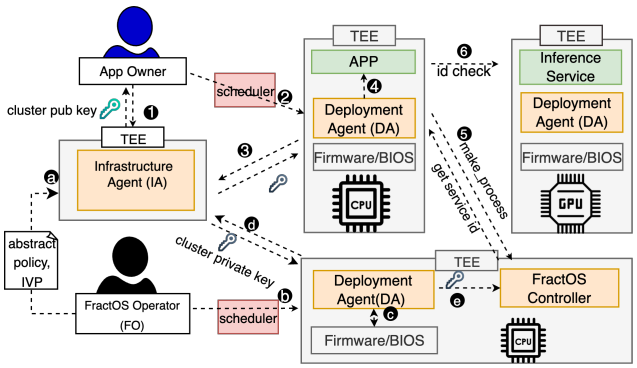


**Figure 2: Steps to attest infrastructure (a–e), and app deployments (1–6)**

## 3.3 Platform Attestation

The platform attestation process consists of five phases, numbered ⓐ–ⓔ in Fig. 2: IA deployment, controller request, platform measurement, platform verification, and controller deployment. ⓐ The FO first deploys an IA inside a TEE, specifying the corresponding abstract policy and controller IVP. ⓑ The FO then encrypts the controller deployment request (which includes the DA configuration) and embeds it into a clear-text request to the cloud scheduler. ⓒ The hypervisor hashes the DA configuration into the immutable host data field of the TEE report. After boot, the TEE collects platform software measurements into an attestation report; i.e., it captures the state of the device and infrastructure software, by hashing them into Intel TDX RTMRs or TPM PCRs [26, 40]. ⓓ The DA sends the attestation report to the IA for verification (see § 3.2). Notably, the IA contains vendor-specific backends to verify heterogeneous hardware roots of trust (see Fig. 1). The IA then checks the measured software states against the abstract policy to ensure conformance with expected configurations. ⓔ If verification succeeds, the IA releases the CPK and the controller IVP to the respective DA. The DA decrypts the deployment request, pulls the controller image, and validates it using the IVP. Once verified, it injects the CPK and environment variables in the deployment request into the controller, enabling secure communication between any attested controllers.

## 3.4 Application Deployment

The app deployment has phases numbered ❶–❻ in Fig. 2. First, ❶ tenants attest the IA, and verify its abstract policy and controller IVP to ensure (in a single step) that the IA, DAs and controllers can be trusted. This facilitates trust across the entire FractOS infrastructure, and returns the cluster public key to the tenant.

After this, tenants can deploy one or more apps using the untrusted cloud scheduler. ❷ The tenant uses the

cluster public key to encrypt and sign the app deployment request, and submits it for deployment. ❸ The DA then authenticates the request received from the scheduler, decrypts the app deployment request using the CPK, calculates the abstract service identity, and performs the relevant co-location checks (see § 3.2). Remember that the DA gets the corresponding CPK only after it is successfully attested against the IA (as in ⓓ earlier); the app deployment integrity and privacy are therefore protected from untrusted components such as the cloud scheduler. ❹ The DA then uses the decrypted app IVP to verify (and optionally decrypt) the integrity of the app image, sends the calculated abstract service identity to the controller assigned to this app, and deploys the app with the inject configuration files and environment variables (including the address of the DA). This identity remains consistent across instances because it only includes relevant deployment parameters; such parameters include a cryptographic hash of any DA and app configuration files, and a set of environment variable values, whose names are provided in the DA configuration. Notably, command line arguments are not considered, due to their order sensitivity, as well as environment variables not present in the DA configuration; it is therefore up to a service client to ensure that the DA configuration used to produce an abstract service identity captures all the relevant configuration files and environment variables that affect the behavior of a deployment. ❺ The app connects with the corresponding controller, and authenticates it using the cluster public key (an untrusted controller cannot have the CPK). As part of this connection, the app passes the address of its DA, which the controller uses to retrieve the abstract service identity for the connecting app.

### 3.5 Service Attestation

An app must attest all services it uses recursively, which is crucial in a public cloud with opaque or dynamic composition (see § 2). ❻ An app attests the services it uses by checking their associated abstract service identity against their expected reference value. This comparison is provided by an *identity-check system call* in the controller, who associates each connection to a service with the identity generated by the DA when deploying it.

Identity checks can therefore be applied recursively, and facilitate using existing (untrusted) cloud schedulers. An app performs an identity check against a particular service instance as part of the connection process, so the assignment can happen via an untrusted component (e.g., an auto-scaling load balancer). In the case of an L7 load balancer, the app must identity-check the load balancer, which in turn must identity-check the service instances it is managing; with a reusable load-balancer, the service identity can be easily embedded as a parameter in the deployment request, therefore enabling recursive composition of identity checks in a safe way.

## 4 Implementation

Our design considers heterogeneous TEEs and secure communications, but the prototype simplifies areas we expect future lift-and-shift solutions to apply, such as secure RDMA, SPDM, and confidential PCIe and CXL [11, 23, 34, 48]. We build an overlay cloud by deploying controllers and apps as confidential containers using Confidential Containers (CoCo) [16]. We chose CoCo for its support of multiple CPU TEEs, secure device I/O, and software infrastructure measurements [14, 30, 42]. Our implementation has two main parts. We need to extend the Key Broker Service [13] to act as the IA, enabling multi-tenant attestation. We also need to deploy apps and controllers using CoCo, implement the DA, and add it into CoCo guest components [15]. This setup ensures DA can be measured using CoCo-provided mechanisms during cVM initialization. The FractOS controller also need to be modified by: (i) extending process creation (`make_process()` call) to retrieve service identities from the DA; and (ii) adding the identity-check system call to allow apps to id-check other services. So far, the DA and the FractOS modifications are completed.

## 5 Evaluation

We conduct a security analysis to examine how well the design mitigates threats from CP, FO, and malicious services, and then benchmark attestation overheads, including service startup and connection.

### 5.1 Security Analysis

The **untrusted CP** could violate FractOS or app deployment policies by allocating a TEE with buggy firmware. However, such attacks are prevented since the IA attests TEE integrity against the abstract policy. If attestation fails, DA in the TEE does not receive the CPK and cannot participate further.

A **malicious FO** may attempt to deploy a compromised controller to exfiltrate tenant secrets. This is not possible in our framework. Upon deployment, DA validates the controller image against the controller IVP. If validation fails, deployment will be aborted, and the corresponding controller instance will not receive the CPK. The IVP is submitted to the IA by the FractOS operator and delivered to DA after platform attestation. Tenants can thus attest the IVP and ensure that only benign controllers are allowed to run.

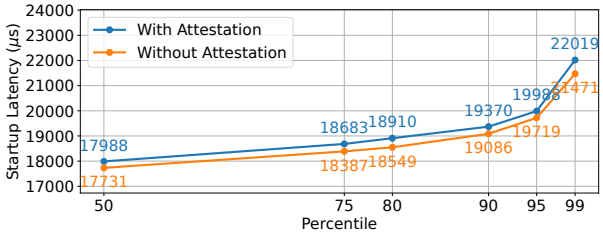


**Figure 3: Process creation latency**

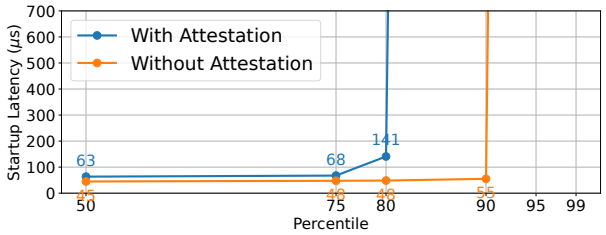


**Figure 4: Service connection latency**

**Malicious service owners** may attempt to trick client apps into using tampered services to extract secrets. For example, a malicious service owner may try to deploy a compromised GPU service that silently leaks patient data. However, such attacks are mitigated by our design. Before any interaction, the hospital app attests the identity of the inference frontend, verifying that it runs the correct binary in a trusted TEE and that it will recursively attest its dependent services as required by the security policy embedded in its identity. If either the frontend or any subservice is compromised, it will be rejected due to an identity mismatch, preserving a trusted chain across the pipeline.

## 5.2 Performance Evaluation

*5.2.1 Methodology.* We evaluate on a 2-node cluster, each with Intel Xeon Silver 4215R CPU, 93 GB DRAM, and a 10 Gbps Mellanox ConnectX-5 NIC. Controllers and the DA run on both nodes. The app is deployed on node 1; the backend service runs on node 2.

*5.2.2 Application Startup Overhead.* We evaluate app startup latency by measuring the time an app takes to complete the `make_process()` call, with and without attestation enabled. Figure 3 presents the `make_process()` latency. At the 50th percentile, the latency increases from 17,731 µs (without attestation) to 17,988 µs (with attestation enabled), and at the 99th percentile, from 21,471 µs to 22,019 µs. Across all percentiles, the overhead remains under 600 µs, representing less than a 3% increase. This overhead arises because the controller must retrieve the caller's ID from the DA when processing the `make_process()`. However, the results show that the attestation introduces minor overhead.

*5.2.3 Service Connection Overhead.* We evaluate the overhead introduced by attestation during service connection. Figure 4 presents the service connection latency. Without attestation, latency remains stable across most percentiles, around 45–55 µs. With attestation enabled, the latency is slightly higher (63–141 µs). This overhead arises from ID verification via RPCs between controllers and the installation of service-specific state (e.g., shared or isolated) during connection setup. The latency impact before saturating the controller is modest (63-141 µs vs. 45-55 µs), but the additional computations done in the controller are sufficient to saturate its worker threads at higher percentiles (80th vs 90th); this is easily solved by adding additional worker threads to the controller.

# 6 Related Work

**Confidential Kubernetes.** Constellation [19] adds k8s scheduler into its TCB, enabling cluster-level attestation but inflating TCB and limiting flexibility. IBM removes the scheduler from the TCB by isolating untrusted Kata agent endpoints [49]. Constellation lacks fine-grained workload attestation, whereas IBM lacks dynamic service composition and recursive attestation, making them unsuitable for our dynamic, multi-party computations.

**Confidential Service Meshes.** Marblerun [45] provides SGX-based confidential scaling with ahead-of-time attestation via a trusted coordinator. Marblerun lacks multi-party and recursive composition, and the coordinator enlarges the TCB attack surface while reducing its flexibility. SCONE [6] uses CAS [22] to attest services and provision TLS certificates for secure point-to-point service connections. However, it does not support dynamic or opaque service composition.

# 7 Conclusions

We introduced scale-out attestation, a novel recursive attestation paradigm that addresses the scalability and trust management challenges in the confidential heterogeneous cloud. We decouple platform and app-level attestation by introducing abstract service identities, delivering scale-out attestation with strong security guarantees while streamlining how trust is established across elastic and opaque service compositions that span multiple trust domains and heterogeneous devices. Our evaluation shows the feasibility of our approach with minimal latency overhead.

# References

[1] Tempus AI. 2024. *Tempus: AI-enabled precision medicine.* https://www.tempus.com

[2] Suzanne Ambiel. 2024. *The Case for Confidential Computing*. https://www.linuxfoundation.org/hubfs/LF%20Research/TheCaseforConfidentialComputing_062724.pdf?hsLang=e

[3] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: Secure and Scalable Aggregate Network Attestation *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 731–742. https://doi.org/10.1145/2976749.2978335

[4] AMD. 2024. *AMD Secure Encrypted Virtualization (SEV)*. https://www.amd.com/en/developer/sev.html

[5] Arm. 2024. *Arm Confidential Compute Architecture (ARM CCA)*. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[6] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 689–703. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[7] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 964–975. https://doi.org/10.1145/2810103.2813670

[8] Azure. 2024. *Introduction to Azure Storage*. https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction

[9] Azure. 2025. *What is Azure Files*. https://learn.microsoft.com/en-us/azure/storage/files/storage-files-introduction

[10] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.* 33, 3, Article 8 (Aug. 2015), 26 pages. https://doi.org/10.1145/2799647

[11] Rob Blankenship and Mahesh Wagh. 2023. *Introducing the CXL 3.1 Specification*. https://computeexpresslink.org/wp-content/uploads/2024/03/CXL_3.1-Webinar-Presentation_Feb_2024.pdf

[12] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. 2017. Lightweight Swarm Attestation: A Tale of Two LISA-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 86–100. https://doi.org/10.1145/3052973.3053010

[13] Confidential Container Community. 2023. *Key Broker Service*. https://github.com/confidential-containers/trustee/tree/main/kbs

[14] Confidential Container Community. 2023. *Trusted Device Manager Architecture*. https://github.com/confidential-containers/guest-components/blob/9de4e6b10af10c25e253c33013d22b2cdaa695e6/tdm/docs/architecture.md

[15] Confidential Container Community. 2024. *Confidential Container Tools and Components*. https://github.com/confidential-containers/guest-components

[16] Confidential Container Community. 2024. *Confidential Containers*. https://github.com/confidential-containers

[17] Confidential Computing Consortium. 2023. *Why is Attestation Required for Confidential Computing?* https://confidentialcomputing.io/2023/04/06/why-is-attestation-required-for-confidential-computing/

[18] Dileep Domakonda. 2025. Secure and Scalable Microservices Architecture: Principles, Benefits, and Challenges. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)* 11, 2 (March-April 2025). https://doi.org/10.32628/CSEIT23112569

[19] Edgelesssys. 2023. *Constellation: Always Encrypted Kubernetes*. https://github.com/edgelesssys/constellation

[20] Mohammad Ewais and Paul Chow. 2024. DDC: A Vision for a Disaggregated Datacenter. arXiv:2402.12742 [cs.AR] https://arxiv.org/abs/2402.12742

[21] Google. 2023. *Verify a Confidential VM instance's firmware (TDX)*. https://cloud.google.com/confidential-computing/confidential-vm/docs/verify-firmware#intel-tdx

[22] Franz Gregor, Wojciech Ozga, Sebastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, Andre Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2020. Trust Management as a Service: Enabling Trusted Execution in the Face of Byzantine Stakeholders . In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, Los Alamitos, CA, USA, 502–514. https://doi.org/10.1109/DSN48063.2020.00063

[23] SPDM Working group. 2023. *Security Protocols and Data Models(SPDM)*. https://www.dmtf.org/standards/spdm

[24] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. 2023. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 419–432. https://www.usenix.org/conference/atc23/presentation/huye

[25] IETF. 2023. Remote ATtestation procedureS (RATS) Architecture. https://www.rfc-editor.org/info/rfc9334.

[26] Intel. 2024. *Intel® Trust Domain Extensions (Intel® TDX)*. https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html

[27] Intel Corporation. 2023. Intel® Software Guard Extensions (Intel® SGX). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html.

[28] Kasun Indrasiri. 2022. Microservices in Practice - Key Achitectural Concepts of an MSA. https://content.wso2.com/wso2/sites/all/images/pdf/microservices-in-practice-key-architectural-concepts-of-an-msa.pdf?utm_source=chatgpt.com.

[29] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2019. Integrating Remote Attestation with Transport Layer Security. arXiv:1801.05863 [cs.CR] https://arxiv.org/abs/1801.05863

[30] Magnus Kulke. 2024. *Building Trust into OS images for Confidential Containers*. https://confidentialcontainers.org/blog/2024/03/01/building-trust-into-os-images-for-confidential-containers/

[31] Dmitrii Kuvaiskii, Dimitrios Stavrakakis, Kailun Qin, Cedric Xing, Pramod Bhatotia, and Mona Vij. 2024. Gramine-TDX: A Lightweight OS Kernel for Confidential VMs. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and*

*Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 4598–4612. https://doi.org/10.1145/3658644.3690323

[32] I-Ting Angelina Lee, Zhizhou Zhang, Abhishek Parwal, and Milind Chabbi. 2024. The Tale of Errors in Microservices. *Proc. ACM Meas. Anal. Comput. Syst.* 8, 3, Article 46 (Dec. 2024), 36 pages. https://doi.org/10.1145/3700436

[33] W. Lindblom. 2022. Evaluation of Security Threats in Microservice Architectures. https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-321085

[34] Rambus Lou Ternullo, Senior Director IP Product Marketing. 2023. *IDE and TDISP: An Overview of PCIe® Technology Security Features*. https://pcisig.com/blog/ide-and-tdisp-overview-pcie%C2%AE-technology-security-features

[35] Boris Lublinsky. 2008. *Service Composition*. Retrieved April 4, 2025 from https://www.infoq.com/articles/lublinsky-soa-composition/

[36] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 412–426. https://doi.org/10.1145/3472883.3487003

[37] Microsoft. 2022. The API gateway pattern versus the Direct client-to-microservice communication. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern.

[38] Nivdia. [n. d.]. *Hopper Confidential Computing: How it Works Under the Hood*. Retrieved April 2, 2025 from https://static.rainfocus.com/nvidia/gtcspring2023/sess/1666639437498001endS/supmat/S51709%20-%20Hopper%20Confidential%20Computing_%20How%20it%20Works%20under%20the%20Hood_1679465925191001GNep.pdf#:~:text=%E2%80%A2%20Requests%20the%20attestation%20report%20from%20the,authenticates%20it%20based%20on%20GPU%20certificate%20chain&text=%E2%80%A2%20Multi%2DInstance%20GPU%20(MIG)%20%E2%80%93%20partitioning%20of%20GPU%20into.

[39] NVIDIA. 2023. *Confidential Computing on NVIDIA H100 GPUs for Secure and Trustworthy AI*. https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/

[40] Reiner Sailer, Leendert Van Doorn, and James P Ward. 2004. *The role of TPM in enterprise security*. Technical Report. Technical Report RC23363 (W0410-029), IBM Research.

[41] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 2004. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (San Diego, CA) *(SSYM'04)*. USENIX Association, USA, 16.

[42] Carlos Segarra, Tobin Feldman-Fitzthum, Daniele Buono, and Peter Pietzuch. 2024. Serverless Confidential Containers: Challenges and Opportunities. In *Proceedings of the 2nd Workshop on SErverless Systems, Applications and MEthodologies* (Athens, Greece) *(SESAME '24)*. Association for Computing Machinery, New York, NY, USA, 32–40. https://doi.org/10.1145/3642977.3652097

[43] Ioannis Sfyrakis and Thomas Gross. 2020. A Survey on Hardware Approaches for Remote Attestation in Network Infrastructures. arXiv:2005.12453 [cs.CR] https://arxiv.org/abs/2005.12453

[44] Yizhou Shan, Will Lin, Zhiyuan Guo, and Yiying Zhang. 2022. Towards a fully disaggregated and programmable data center. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems* (Virtual Event, Singapore) *(APSys '22)*. Association for Computing Machinery, New York, NY, USA, 18–28. https://doi.org/10.1145/3546591.3547527

[45] Edgeless system. 2024. *MarbleRun: a framework for creating distributed confidential computing apps*. https://github.com/edgelesssys/marblerun

[46] Edgeless system. 2024. *Remote Attestation*. https://www.edgeless.systems/wiki/what-is-confidential-computing/remote-attestation#:~:text=Remote%20attestation%20is%20a%20crucial,is%20addressed%20by%20remote%20attestation

[47] Rajat Tandon. 2020. A Survey of Distributed Denial of Service Attacks and Defenses. arXiv:2008.01345 [cs.CR] https://arxiv.org/abs/2008.01345

[48] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. 2020. sRDMA – Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 691–704. https://www.usenix.org/conference/atc20/presentation/taranov

[49] Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Christophe de Dinechin, Pau-Chen Cheng, and Hani Jamjoom. 2024. Crossing Shifted Moats: Replacing Old Bridges with New Tunnels to Confidential Containers. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 1390–1404. https://doi.org/10.1145/3658644.3670352

[50] Lluís Vilanova, Lina Maudlej, Shai Bergman, Till Miemietz, Matthias Hille, Nils Asmussen, Michael Roitzsch, Hermann Härtig, and Mark Silberstein. 2022. Slashing the disaggregation tax in heterogeneous data centers with FractOS. In *European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3492321.3519569

[51] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 681–696. https://www.usenix.org/conference/osdi18/presentation/volos

[52] Samuel Wedaj, Kolin Paul, and Vinay J. Ribeiro. 2019. DADS: Decentralized Attestation for Device Swarms. *ACM Trans. Priv. Secur.* 22, 3, Article 19 (July 2019), 29 pages. https://doi.org/10.1145/3325822

[53] Wikipedia. 2025. *Transport Layer Security*. https://en.wikipedia.org/wiki/Transport_Layer_Security

[54] Mikko Ylinen and Dr. Malini Bhandaru. 2024. *Confidential Cloud Native Attestation Challenges and Opportunities, OC3 2024*. https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction

[55] Yan Zhai, Qiang Cao, Jeffrey Chase, and Michael Swift. 2017. TapCon: Practical Third-Party Attestation for the Cloud. In *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/hotcloud17/program/presentation/zhai

[56] Mark Zhao, Mingyu Gao, and Christos Kozyrakis. 2022. ShEF: shielded enclaves for cloud FPGAs. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 1070–1085. https://doi.org/10.1145/3503222.3507733

[57] Yongbin Zhou and Dengguo Feng. 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. *IACR Cryptol. ePrint Arch.* 2005 (2005), 388. https://api.semanticscholar.org/CorpusID:9365379