DICE-anchored Realms on Arm CCA

Lorenzo Ferro[1,*], Silvia Sisinni[1], Flavio Ciravegna[1], Enrico Bravi[1] and Antonio Lioy[1]

[1]Politecnico di Torino, Dipartimento di Automatica e Informatica (DAUIN), Torino, 10129, Italy

Abstract

Confidential computing on edge and embedded platforms increasingly requires isolation mechanisms that support mutually distrusting workloads and provide verifiable evidence of correct execution. Arm's Confidential Compute Architecture (CCA) introduces hardware-protected Realms and an attestation framework that reports measurements of platform and Realm state. However, CCA offers no means to derive application identities or cryptographic keys bound to measured software, limiting its integration with infrastructures that rely on measurement-based identity and complicating scalable trust establishment. In contrast, the Device Identifier Composition Engine (DICE) specification defines a mechanism for deriving keys from a device-rooted secret and the measurements of successive boot stages, enabling implicit attestation: the ability to infer execution integrity from key use alone. This paper presents an architecture that integrates DICE-based derivation into the CCA firmware stack and Realm-initialisation process. The design produces Realm-specific keys that depend on both the hardware root of trust and the measured Realm image, enabling implicit attestation while remaining compatible with CCA's explicit attestation tokens. This integration strengthens the trust guarantees available to confidential workloads and allows CCA devices to participate in ecosystems that employ DICE for supply-chain validation and policy enforcement. A case study on confidential machine-learning inference at the edge illustrates the practical implications of this design. It shows how DICE-anchored Realms can protect model weights, prompts, and inference outputs while providing verifiable evidence of their execution context. The combined use of CCA isolation and DICE-derived identities offers a scalable environment for trustworthy multi-tenant computation on Arm-based systems.

Keywords

ARM, ARM-CCA, TEE, Confidential-AI, DICE, Edge ML

1. Introduction

Edge devices increasingly undertake operations that require strict confidentiality. Industrial gateways perform anomaly detection on production telemetry before forwarding aggregated summaries to the cloud. Home-automation hubs infer personal routines from sensor data to optimise energy consumption. Medical wearables analyse physiological signals in real time to trigger early-warning alerts. In these settings, computation occurs outside physically protected environments and often under the control of the device owner rather than the data owner. As a result, several mutually independent and sensitive workloads coexist on the same hardware platform, a pattern that characterises many modern multi-tenant edge deployments [1]. Recent work on embedded and edge machine-learning systems has shown that even relatively small models can leak private information or expose cross-tenant side channels when isolation is insufficient [2, 3, 4, 5]. At the same time, studies of industrial and Internet of Things (IoT) execution platforms emphasise that gateways consolidating workloads for different principals require strong separation to mitigate interference and misconfiguration [6, 7, 8, 9, 10].

Guaranteeing the confidentiality of such workloads requires more than protecting data in transit or at rest: the execution environment itself must be capable of preventing co-resident software from observing or influencing sensitive computations, and must allow remote stakeholders to establish trust in the behaviour of deployed applications. The Arm ecosystem has traditionally relied on TrustZone

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT

*Corresponding author.

†These authors contributed equally.

lorenzo.ferro@polito.it (L. Ferro); silvia.sisinni@polito.it (S. Sisinni); flavio.ciravegna@polito.it (F. Ciravegna); enrico.bravi@polito.it (E. Bravi); antonio.lioy@polito.it (A. Lioy)

0009-0009-3286-7366 (L. Ferro); 0000-0003-1870-6303 (S. Sisinni); 0009-0006-6668-8022 (F. Ciravegna); 0009-0006-1832-0274 (E. Bravi); 0000-0002-5669-9338 (A. Lioy)

for this purpose, yet TrustZone was designed around a coarse two-world split between "secure" and "normal" software, making it unsuitable for hosting several mutually distrusting secure applications. As documented in architectural analyses and surveys of TrustZone, this model provides isolation at the granularity of a trusted operating system rather than at the level of individual secure applications, limiting its suitability for scenarios involving mutually distrusting components [11, 12, 13].

Arm's Confidential Compute Architecture (CCA) was introduced to address these limitations by enabling applications to run in hardware-isolated "Realms" that are protected even from privileged host software. CCA extends traditional Arm architectures with a dedicated execution world for confidential workloads, managed by the Realm Management Extension (RME) and its associated firmware components [14]. This design explicitly addresses multi-tenant Confidential Computing: several isolated applications can coexist on the same System-on-Chip (SoC), each separated from the hypervisor and from other Realms. CCA also provides an attestation mechanism through which a remote party can verify the integrity of the platform and the initial state of a Realm. This attestation model has been formalised in the Realm Management Monitor (RMM) architecture specification and in Arm's contribution to the IETF Remote ATtestation procedureS (RATS) working group, which defines a CBOR-encoded CCA attestation token [15, 16].

Yet, the attestation mechanism currently associated with CCA remains predominantly explicit: a Realm requests an attestation report, the platform signs it, and a verifier evaluates the claims. This model is effective for verifying the state at a specific moment, but does not provide an implicit link between application-level keys and the underlying platform state. In contrast, the Device Identifier Composition Engine (DICE) architecture, codified by the Trusted Computing Group (TCG) and adopted in domains ranging from IoT devices to Android's security foundations, defines a mechanism for deriving cryptographic identities from measurements taken across the boot chain [17, 18]. DICE-derived keys inherently embody the state of the code that produced them, enabling implicit attestation, where the successful use of a key already attests to the integrity of the execution context. This capability allows trust establishment to be integrated more directly into application-level protocols, without requiring explicit attestation exchanges for every decision.

The absence of such implicit attestation semantics in CCA leaves several practical and conceptual gaps. Realms lack a standardised way to obtain identity keys tied to their code and to the specific platform on which they run. Remote verifiers must rely exclusively on explicit attestation flows, which complicates orchestration at scale. Additionally, CCA devices cannot easily be integrated into broader trust infrastructures that already rely on DICE-based identity hierarchies for supply-chain validation, configuration assessment, and distributed policy enforcement.

This paper proposes an architecture that addresses these limitations by binding CCA's isolation model to the DICE identity and key-derivation framework. Instead of treating CCA attestation as a standalone mechanism, we extend the CCA firmware stack and Realm creation process by introducing a sequence of DICE-compliant identity-derivation steps. The result is a model in which Realms acquire identities and cryptographic material that are automatically tied to their measured state and to the device on which they execute, supporting both explicit attestation (compatible with Arm's existing token format) and implicit attestation (in line with the semantics expected by modern secure protocols). By grounding CCA Realms in a widely supported identity standard, this approach enables portable verification across heterogeneous systems, supports scalable multi-tenant deployments, and simplifies trusted key derivation for confidential workloads at the edge [19].

The remainder of this paper is organised as follows. Section 2 provides background on Trusted and Confidential Computing, covering Remote Attestation (RA), Trusted Execution Environments (TEEs), Arm TrustZone, the Arm CCA, and the DICE framework. Section 3 introduces the threat model and the design goals that guide our proposed integration. Section 4 details the system architecture for DICE-anchored Realms, outlining how Realm's identity derivation and attestation semantics are embedded into the CCA firmware and Realm lifecycle. Section 6 presents a case study demonstrating the applicability of the approach to confidential machine-learning workloads at the edge, using CCA's isolation and DICE-derived identities to secure inference operations. Section 7 surveys related work across TEE architectures, attestation frameworks, and prior efforts to integrate DICE into heterogeneous

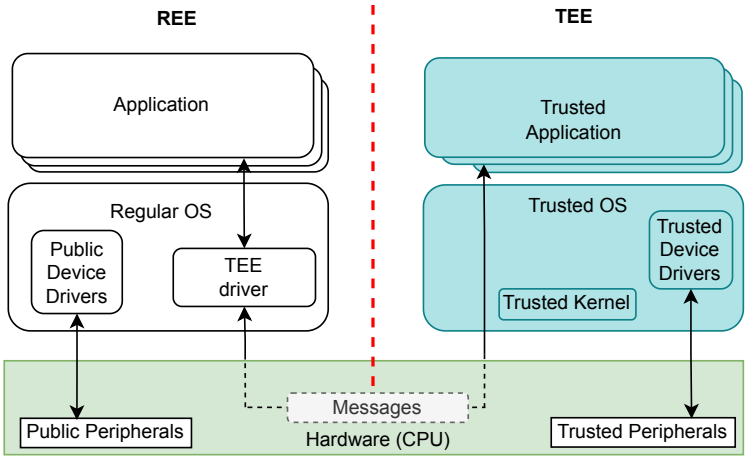

Figure 1: TEE isolation architecture [24].

platforms. Section 8 discusses open challenges and avenues for future refinement and validation, while Section 9 concludes the paper.

2. Background

2.1. Trusted and Confidential Computing

Trusted and Confidential Computing defines a set of security principles designed to protect the integrity, confidentiality, and authenticity of computation, even in the presence of a powerful adversary. These paradigms address the growing need to execute sensitive workloads on platforms that may not be fully trusted, such as shared, virtualised, or cloud systems.

At a high level, Trusted Computing focuses on ensuring that a system behaves as expected [20]. This includes establishing a Root of Trust (RoT), which is a component in charge of exposing trustworthy cryptographic primitives in a system. This includes measurement, secure storage, reporting, and verification. In contrast, Confidential Computing extends these guarantees by protecting data and code while they are in use, preventing unauthorised access even from privileged system software such as the operating system or hypervisor [21]. This is typically achieved by using TEEs.

Together, they enable verifiable and privacy-preserving computation across a wide range of deployment scenarios [22]. Trusted computing mechanisms provide the foundation for verifying the platform state, while Confidential Computing technologies isolate sensitive workloads and enforce strong execution guarantees.

In this context, assessing and verifying the trustworthiness of a remote platform becomes essential. RA plays a central role in bridging Trusted and Confidential Computing by allowing external entities to obtain cryptographic evidence about the system's state before interacting with it or provisioning sensitive data [23].

2.2. Trusted Execution Environments

TEEs technologies have the purpose to provide a shielded and isolated environment to execute critical applications. They are typically implemented in hardware as Central Processing Unit (CPU) security extensions to provide hardware memory and execution isolation. The standard architecture described by the Global Platform association [24] represents a platform composed of two execution environments: a Rich Execution Environment (REE) and a TEE (Figure 1). The REE is the "normal" execution environment, not protected or isolated, where the non-sensible applications are executed. The TEE is the hardware-

protected environment, protected and isolated from the REE. Depending on the specific TEE technology, the isolation can be only between the REE and the TEE (e.g., , ARM TrustZone [11]), or there can also be isolation among the different applications (Trusted Applications) running in the TEE (e.g., Keystone [25]).

Given the strong isolation among several components, the communication mechanism needs to be defined accordingly to meet the security requirements. The communication between the REE and the TEE is managed by a low-level firmware, which is typically the most privileged one, and it is in charge of switching the context of execution when passing from the REE and the TEE in order to guarantee the protection.

2.2.1. Arm TrustZone

Arm TrustZone is a hardware-based security technology that provides system-wide isolation on Arm platforms [12]. It partitions the system into two isolated execution environments, referred to as the *Normal world* and the *Secure world*, which are enforced through hardware mechanisms at the processor and memory subsystem levels.

The Normal world hosts the rich operating system and non-sensitive applications, while the Secure world is reserved for trusted software components. Code executing in the Secure world can access both secure and non-secure resources, whereas software in the Normal world is prevented from directly accessing secure memory and peripherals. Transitions between the two worlds are tightly controlled and mediated by privileged firmware.

TrustZone mainly serves as a foundational building block for implementing TEEs on Arm platforms. Recent architectures, such as CCA, build upon and extend the TrustZone model to provide stronger confidentiality guarantees for complex and virtualised workloads.

2.2.2. Arm Confidential Compute Architecture

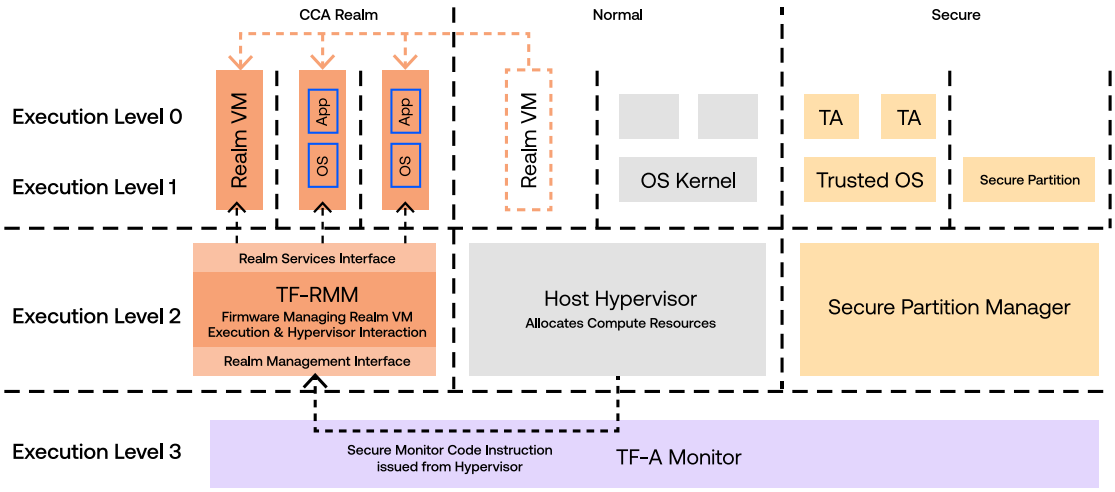

Figure 2: Arm Confidential Compute Architecture [26].

Arm CCA is a security architecture designed to enable Confidential Computing on Arm-based platforms. It defines three primary execution states (Fig. 2): Normal, Secure, and Realm worlds. The *Normal world* hosts non-confidential workloads and system software, including the host hypervisor. The *Secure world* executes trusted services and builds upon the existing Arm TrustZone architecture. The *Realm world* is used to run isolated enclaves, referred to as Realms. Transitions between the Normal, Secure, and Realm worlds are mediated by trusted firmware operating in a dedicated *Root world*. This

functionality is provided by the Trusted Firmware-A (TF-A) Monitor, which executes at the processor RoT and enforces secure world switching and isolation.

Within the Realm world, the execution and lifecycle of Realms are managed by the Realm Management Monitor (RMM). The RMM responds to requests from the hypervisor running in the Normal world and coordinates Realm operations such as context switching and memory assignment. Communication between the hypervisor and the RMM occurs through the TF-A Monitor, which controls transitions between the Realm Physical Address Space (PAS) and the Normal PAS.

While the RMM enforces isolation and manages Realm execution, it does not implement system-wide resource management policies. Decisions such as scheduling and memory allocation remain under the control of the host hypervisor, preserving its role as the system resource manager.

2.3. DICE

The DICE framework [18] enables the construction of a layered architecture, where each layer builds upon the trusted guarantees provided by the lower ones (see Fig. 3). This allows to generate specific identifiers for each component of the Trusted Computing Base (TCB), which is the set of hardware, firmware and software elements critical for the system's security [27, 28].

DICE acts as a lightweight RoT for platforms that do not include a Trusted Platform Module (TPM), most commonly due to hardware and resource constraints. Such conditions are typical in IoT, embedded, and edge devices. In the context of this work, the DICE specifications are particularly relevant, as they provide a practical means to establish strong cryptographic identities and attestation capabilities.

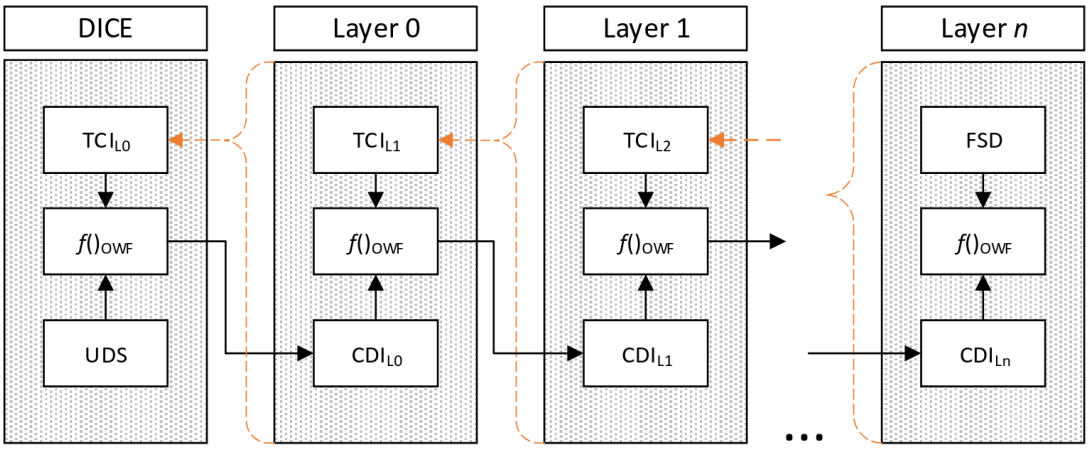

Figure 3: The DICE Layered Architecture [18].

At a high level, the DICE architecture allows each layer to derive cryptographic key pairs used to identify each layer and attest its integrity. These keys are derived from the Compound Device Identifier (CDI), which binds together two elements: a secret inherited from the previous layer and the TCB Component Identifier (TCI), which is a measurement of the next layer to be executed. For a generic layer N, the CDI is computed as

$$CDI_N = F(CDI_{N-1} \parallel TCI_N),$$

where CDI_{N-1} represents the CDI of the previous layer, TCI_N is the measurement of the next layer, and F is a state-of-the-art cryptographic one-way function. The DICE core layer is an exception. Since no prior CDI exists, the initial CDI is computed as

$$CDI_{CORE} = F(UDS \parallel TCI_{CORE})$$

where the Unique Device Secret (UDS) is a unique, manufacturer-provided value stored in a secure read-only memory accessible only to the DICE Core layer. Because of how CDIs are created, if any component in the chain is modified or replaced, its TCI will change, and as a result, all CDIs from that layer onward will also change.

3. Threat Model and Design Goals

We envision deploying DICE-anchored Realms on edge devices operating within unprotected external networks. In these environments, the device is physically accessible, and the underlying hosting infrastructure is inherently prone to compromise. Therefore, ensuring workload confidentiality and integrity requires a security architecture that explicitly assumes a great part of the system software, including the operating system and hypervisor, as potentially hostile.

3.1. Trust Assumptions

To define the security boundaries of our system, we identify the minimal set of components that must be trusted to uphold the DICE-anchored guarantees:

- **The DICE Hardware RoT:** We assume the immutable hardware mechanism, such as the Boot Read Only Memory (ROM), correctly protects the UDS and accurately computes the initial CDI (CDI_0). The UDS is assumed to be inaccessible to any software layer.
- **Arm CCA Hardware Primitives:** We rely on the correctness of the RME hardware, specifically the Granule Protection Table (GPT) checks, to enforce memory isolation between the Normal, Secure, and Realm worlds.
- **Trusted Firmware (TCB):** The TF-A Monitor and the RMM form the static TCB. We assume these components provide correct isolation enforcement capabilities, and correctly implement the derivation steps to pass CDIs up the DICE chain.

3.2. Assumptions and Scope

While we assume the RMM and TF-A Monitor are formally verified or, at least, extensively audited, we adopt a realistic stance regarding the Realm workloads. In this case, the application logic within the Realm may contain vulnerabilities.

To clearly delineate the scope of this work, we explicitly exclude:

- **Denial-of-Service:** As the Hypervisor controls the physical CPU scheduling, it can deny execution time to the Realm. We consider availability out of scope for this architecture.
- **Physical Attacks:** We assume the attacker does not have the equipment to physically damage the chip or probe internal buses, although RME's memory encryption provides defense against cold-boot attacks and passive bus snooping.
- **Micro-architectural Side-Channels:** Sophisticated timing or speculative execution attacks (e.g., Spectre-class exploits) against the isolation boundary are considered out of scope.

4. System Design: DICE-anchored Realms

The design presented in this section integrates the TCG DICE Layering Architecture [18] into the Arm CCA as specified by the RME [14] and the RMM [15]. This integration provides Realms with load-time cryptographic identities that capture the complete measurement sequence (i.e., from immutable hardware through firmware, the RMM, hypervisor Realm configuration, and the Realm's initialisation state itself), while preserving full compatibility with Arm's architectural requirements, including the stringent Hardware Enforced Security (HES) properties defined in RME [14, B4.1.5].

The approach exploits the architectural alignment between Arm's attestation model and DICE's identity derivation semantics. Arm CCA's attestation mechanism already defines the Realm Initial Measurement (RIM), Realm Extensible Measurements (REMs), Realm configuration metadata, and the generation of the Realm Attestation Key (RAK) [15, A7.1–A7.2]. DICE enriches these explicit claims with an implicit identity chain, wherein each mutable software layer derives a cryptographic identifier (CDI) from its predecessor and its measured state. The resulting hierarchy strengthens the trust and provenance evidence available to relying parties while maintaining perfect interoperability with existing CCA systems.

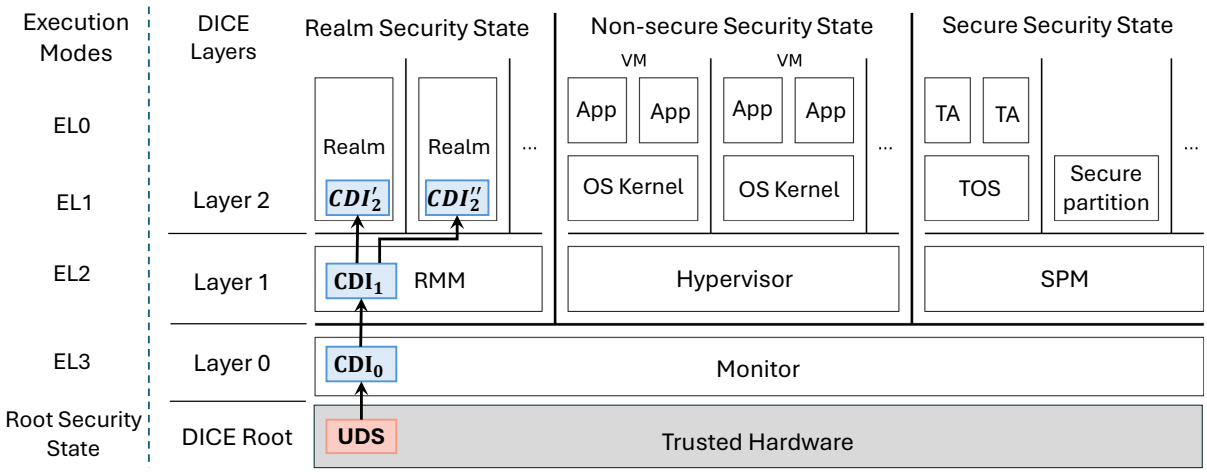

Figure 4: System Software Architecture with DICE.

4.1. DICE Layering Mapped to the CCA Firmware Stack

4.1.1. DICE Root: Hardware Root of Trust

The DICE model begins with an immutable trusted hardware component, the DICE Root, which encapsulates a UDS, as described in Section 2.3. The UDS must remain exclusively accessible to this immutable component, never disclosed to any mutable software, and never stored in persistent memory accessible after reset [17]. On Arm platforms, this immutable trust anchor is naturally instantiated by the boot ROM, which performs the secure-boot authentication of the TF-A according to RME [14, B9] and may access provisioning data stored in Root Non-Volatile Storage (RNVS) as described in RME [14, B4.1.1]. RNVS itself is not the repository for the UDS; instead, it provides confidential parameters that the boot ROM may internally combine with hardware logic to reconstruct or derive UDS-related root secrets (e.g., Physical Unclonable Function (PUF)-related metadata). Since the UDS must never be exposed to mutable software, the boot ROM implements the complete computation of CDI_0 internally: after authenticating the Monitor (typically implemented by TF-A), it computes CDI_0 as a function of the UDS and the Monitor's measurement (TCI_0), and passes CDI_0 to the Monitor only once during handoff:

$$CDI_0 = F(UDS \,||\, TCI_{Monitor})$$

The ROM exposes no callable interface, thereby preventing any mutable layer from re-deriving or abusing UDS-derived material. This behaviour is strictly consistent with both DICE non-extractability requirements and the HES constraints of RME [14, B4.1.5]. Once the Monitor receives CDI_0, all subsequent layers of the CCA initialisation process align naturally with the successive DICE layers.

4.1.2. DICE Layer 0: Monitor

The first mutable layer in the CCA firmware stack is the Monitor, which executes at Exception Level 3 (EL3) within the Root Security State. This privileged component is responsible for establishing and maintaining the separation among the Normal, Secure, and Realm worlds, mediating transitions across Protection State (PS) boundaries, and enforcing the Root world's integrity guarantees as defined in the RME architecture [14]. Because the Monitor executes at the boundary between immutable hardware and all subsequent mutable software, it constitutes the first DICE layer above the hardware root. In the DICE hierarchy, the Monitor receives the initial CDI, CDI_0, computed by the immutable boot ROM from the UDS and the Monitor's TCI, $TCI_{Monitor}$. The ROM transfers CDI_0 to the Monitor exactly once during early boot, and no interface exists for later re-derivation.

At this stage, the Monitor becomes responsible for deriving the next layer's identity, CDI_1, which corresponds to the RMM. Consistent with the DICE Layering Architecture [], the Monitor must first

measure the RMM image and its configuration parameters before handing execution to it. These measurements include the executable image, the RMM's architectural feature set, interface versioning relevant to Realm Management Interface (RMI) calls, and any implementation-specific configuration metadata that affects its security-relevant behaviour. We denote the resulting measurement as: TCI_{RMM}. Using this measurement, the Monitor computes:

$$CDI_1 = F(CDI_0 \parallel TCI_{RMM})$$

where F is the same cryptographic one-way function family used in prior stage (e.g., HMAC-based Key-Derivation Function (KDF)). The Monitor then conveys CDI_1 to the RMM through the protected transition into the Realm Management Security Domain (RMSD).

The RMSD is a logically isolated firmware domain formally defined in the RME architecture [14]. It encompasses the execution of the RMM at Realm Exception Level 2 (R-EL2) and is granted exclusive access to Realm PAS resources and Realm measurement structures. The RMSD cannot be accessed by the Normal or Secure worlds, and only EL3 transitions controlled by the Monitor can enter it. This design ensures that the Monitor and RMM form a tightly isolated two-layer TCB above the hardware, directly corresponding to the first two mutable layers of a DICE chain. By deriving and passing CDI_1 to the RMM in this controlled manner, the Monitor anchors Realm identity in a measurement-based lineage that begins in hardware and continues through the firmware responsible for establishing the Realm world.

4.1.3. DICE Layer 1: Realm Management Monitor (RMM)

The RMM is the central firmware component responsible for instantiating, managing, and attesting Realms. It executes at R-EL2 inside the RMSD, a protection domain strictly enforced by RME's GPT checks and memory-access control rules [14]. The RMSD provides the RMM with exclusive ownership of Realm PAS mappings and complete isolation from the Normal and Secure worlds, including the host hypervisor. Within this domain, the RMM orchestrates Realm lifecycle operations (i.e., creation, initialisation, execution management, and teardown) while ensuring that Realm confidentiality is preserved.

Upon receiving CDI_1 from the Monitor, the RMM becomes responsible for extending the DICE trust chain into the Realm world. This requires defining and computing the Realm's TCI, which reflects the Realm's RIM, configuration state, and additional REMs as defined in the RMM specification [15, A7.1]. The RMM accumulates the following elements:

1. the measurement of the Realm's executable image and initial data pages supplied via RMI calls;
2. architectural parameters such as Intermediate Physical Address (IPA) size, Generic Interrupt Controller (GIC) configuration, and whether Large Physical Addressing v2 (LPA2) or other optional features are enabled;
3. the Realm Personalization Value (RPV), which ensures that distinct Realms executing identical binaries receive distinct identities;
4. other security-relevant configuration supplied through the hypervisor-mediated Realm creation interface.

These values collectively define the Realm's TCI, denoted: $TCI_{Realm}(R)$. Following DICE semantics, the RMM derives a Realm-specific CDI:

$$CDI_2(R) = F(CDI_1 \parallel TCI_{Realm}(R))$$

This derivation ensures that the Realm's identity depends not only on the correctness of the hardware and Monitor stages, but also on the exact initial state and configuration of the Realm itself. Because all derivation logic executes within the RMSD, the resulting $CDI_2(R)$ remains inaccessible to the hypervisor, the Secure world, or the Realm.

The RMM then uses $CDI_2(R)$ internally to derive the RAK. The private RAK never leaves the RMSD; the RMM exposes only the public portion through the attestation interface, maintaining full compatibility with CCA's explicit attestation model [15, A7.2]. In parallel, the RMM derives a Realm Identity Key (RIK) following the identity-key conventions of IEEE 802.1AR [29], which defines long-lived asymmetric key pairs used to authenticate devices during secure communication protocols. However, unlike the 802.1AR model, where identity keys are provisioned as static credentials bound to a hardware device, the RIK in our architecture is derived from $CDI_2(R)$, thereby inheriting the DICE semantics that bind cryptographic identity to the measured state of both the platform and the Realm. This ensures that the RIK serves as a stable, Realm-scoped device identity suitable for Internet secure protocols (e.g., signing opaque challenges, establishing mutually authenticated channels), while simultaneously guaranteeing that the key is implicitly anchored in the Realm's measurement chain and cannot be migrated or replayed on a different platform or under a different Realm configuration.

By generating the RIK through DICE-based derivation rather than static provisioning, the Realm obtains a secure-communication identity that is cryptographically tied to its initialisation state, execution context, and the underlying hardware RoT. The combined use of $CDI_2(R)$, RAK, and RIK therefore enables Realms to participate in attestation flows that support both explicit, evidence-carrying claims (via the RAK) and implicit identity guarantees for secure session establishment (via the RIK), integrating CCA Realms into existing trust infrastructures, while remaining compliant with the architectural constraints of Arm RME and RMM.

4.1.4. DICE Layer 2: Realm Instance

The Realm Instance constitutes the execution environment for confidential workloads at Realm Exception Level 1 (R-EL1) and Realm Exception Level 0 (R-EL0). At this stage, the DICE chain terminates: $CDI_2(R)$ embodies the complete sequence of measurements, from immutable hardware through firmware and the RMM, together with the Realm's initial configuration. Although Realms never access $CDI_2(R)$, the private RAK, or any other secret material confined within the RMSD, they rely on the cryptographic identities derived from it to authenticate their execution context to external entities.

From an architectural standpoint, the Realm layer realises three key properties. First, the Realm's identity is uniquely bound to its measured initial state. Any deviation in the Realm's image, configuration parameters, or RPV yields a different $TCI_{Realm}(R)$ and therefore a different $CDI_2(R)$ and associated key hierarchy. This enforces DICE's non-forging guarantees in the CCA setting. Second, Realms operate entirely within the Realm PAS. GPT checks strictly prevent their memory from being mapped into the Normal or Secure worlds, ensuring that the provenance captured by $CDI_2(R)$ at creation-time remains intact throughout execution.

The RMM exposes two distinct interfaces that allow the Realm to use the keys derived from $CDI_2(R)$ without disclosing their private parts.

Attestation interface (RAK). Through the Realm Services Interface (RSI), the Realm may request attestation evidence. The RMM uses the RAK (deterministically derived from $CDI_2(R)$ in our architecture) to sign an attestation token containing the RIM, REMs, configuration descriptors, and other architecturally mandated claims [15, A7.2]. The signed token is returned to the Realm and can be forwarded to relying parties in the Normal world. Because RAK_{priv} never leaves the RMSD, the Realm obtains verifiable attestation without acquiring sensitive key material.

Identity interface (RIK). In addition to attestation, the Realm may require a persistent, externally verifiable identity for secure communication protocols that follow the semantics of IEEE 802.1AR [29]. For this purpose, the RMM exposes a controlled operation through which the Realm can request RIK-backed cryptographic actions (e.g., signing an opaque challenge or signing an ephimeral public key for enrollment). The RIK is derived from $CDI_2(R)$ using the same DICE lineage semantics as the RAK; consequently, it implicitly encodes both the platform's hardware identity and the Realm's measured state. Only the RIK's public portion is made available to the Realm, while all private-key operations occur

within the RMSD. This design preserves 802.1AR's usage model for device identity keys but replaces static provisioning with measurement-derived identity, ensuring non-transferability, non-replayability, and binding to the Realm's initialisation.

Realms may also implement internal key hierarchies or invoke application-level DICE constructs, but these remain orthogonal to the architectural chain defined above and do not affect the RAK, RIK, or the attestation semantics mandated by the RMM specification.

5. Implementation

The architectural DICE hierarchy described in Section 4 admits multiple implementation strategies, each preserving the invariant trust chain:

$$UDS \rightarrow CDI_0 \rightarrow CDI_1 \rightarrow CDI_2(R)$$

while differing in how CDIs and derived keys are handled at runtime. The choice of implementation affects the confidentiality properties and operational flexibility of CDI-dependent secrets, including both the RAK and the RIK. In every realisation, however, the RME protection domains guarantee that no CDI or CDI-derived secret becomes accessible outside the RMSD.

5.1. Direct CDI Handling within Firmware

In the most straightforward implementation, the immutable boot ROM computes CDI_0 internally and transfers it to the Monitor during the secure boot handoff. The Monitor stores CDI_0 in Root PAS memory and uses it to compute CDI_1 after measuring the RMM image and configuration. Upon entering the RMSD, the Monitor passes CDI_1 to the RMM, which retains it in shielded memory regions protected by GPT rules.

When the hypervisor initiates Realm creation, the RMM evaluates the Realm's TCI and derives $CDI_2(R)$. This CDI serves as the seed for all Realm-scoped cryptographic identities. Through internal KDFs, the RMM computes both the RAK and the RIK:

- The RAK is used exclusively for attestation. All signing operations take place within the RMSD; realms receive attestation tokens, not the private RAK.
- The RIK is used to represent the Realm as a secure communication endpoint, following IEEE 802.1AR identity semantics but relying on DICE measurements rather than static provisioning. The RMM exposes the RIK's public key to the Realm and permits RIK-backed signatures via controlled RSI calls, ensuring that RIK_{priv} remains confined.

After key derivation, the RMM may erase $CDI_2(R)$ if the implementation chooses to retain only the derived keys or to rely on deterministic regeneration during attestation operations.

5.2. CDI Confinement via a DICE Protection Environment

A second implementation strategy employs a DICE Protection Environment (DPE) [30], which encapsulates CDIs and their derivation chain entirely within a dedicated hardware secure subsystem. Rather than receiving raw CDIs, firmware components interact with the DPE through opaque handles:

1. The ROM invokes `DPE.Initialise` to create the root context from the UDS. The resulting handle corresponds to CDI_0, though its value remains inaccessible.
2. The Monitor requests a derived context for the RMM measurement, producing a handle representing CDI_1.
3. The RMM, in turn, requests a Realm-specific context based on $TCI_{Realm}(R)$, obtaining an opaque $CDI_2(R)$ handle.

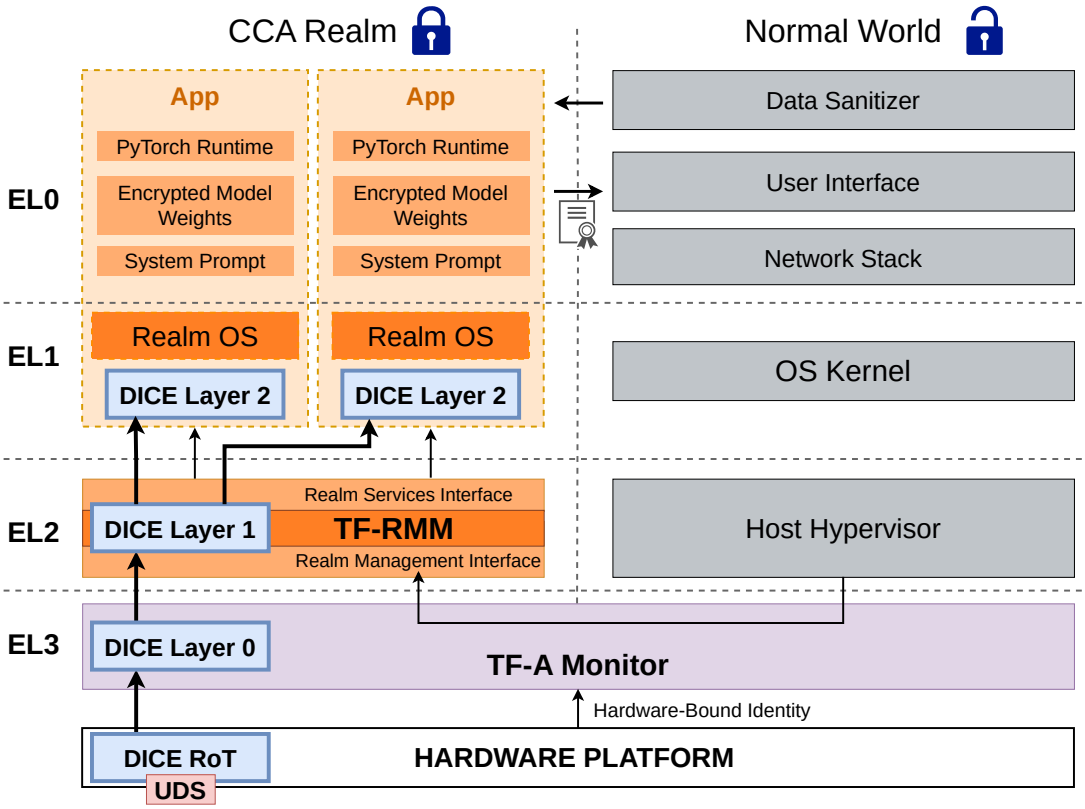

Figure 5: Architecture for ML integrated in ARM CCA with DICE.

Within this model, all RAK and RIK derivations are performed through DPE operations such as `DeriveKey` or `Sign`, ensuring that private key material is never exposed to firmware. The RMM still controls the invocation of these operations on behalf of the Realms, allowing it to enforce the CCA execution model and the attestation semantics defined in the RMM specification [15]. The Realm continues to receive only the public identity (RIK_{pub}) and RAK-signed attestation tokens. This approach offers stronger resilience against firmware compromise and persistent state attacks.

6. Case Study: Confidential Machine Learning at the Edge

To demonstrate the practical utility of integrating CCA with DICE, we examine a deployment of Confidential Machine Learning (ML) at the edge. Hosting an entire large model stack within a TEE is often resource-prohibitive due to memory constraints. Therefore, we propose a hybrid architecture that leverages the dynamic partitioning capabilities of CCA to optimise both performance and security. In this model, the untrusted OS handles non-sensitive workloads such as user interactions, network stack management, and initial data pre-processing. This architecture allows input data to be sanitised in the normal world before it ever crosses the security boundary into the trusted environment. The Realm is reserved exclusively for security-critical components, hosting the PyTorch runtime instance, proprietary model weights, private cryptographic keys, and the sensitive initial system prompt (Fig. 5).

To facilitate the interaction between the untrusted OS and the protected Realm without compromising isolation, we utilise a distinct shared memory region designated as non-secure by the RMM. Technically, this is achieved by configuring the GPT to assign the buffer's physical pages to the Non-Secure Physical Address Space (NS-PAS). While the Normal World is strictly barred from accessing the Realm's private encrypted memory, the CCA architecture permits the Realm to map and read pages from the NS-PAS. This asymmetric visibility creates a controlled 'drop-off zone' accessible to both worlds. The workflow

proceeds as follows: the Normal World application writes pre-processed input tensors into this shared buffer and signals the Realm via an SMC. Upon receiving the signal, the Realm maps the buffer, performs inference using the isolated PyTorch runtime, and writes the results back to the shared memory. This "zero-copy" approach minimises the overhead typically associated with world-switching in TEEs, ensuring that latency remains within acceptable limits for real-time edge applications.

This architectural separation establishes a robust defence against specific edge-based attack vectors through strict resource isolation. By sequestering the model execution, we ensure that inference resources cannot be contaminated or poisoned by untrusted data sources residing in the potentially compromised rich OS. Even if an attacker gains root access to the normal world, the memory encryption inherent to the Realm prevents them from inspecting the model configuration or extracting intellectual property. Furthermore, this isolation guarantees prompt confidentiality; the initial system prompt remains resident solely in encrypted memory. It is never disclosed to the OS or device owner, ensuring the model logic adheres to its intended guardrails without interference.

The integration of DICE is fundamental to transforming this isolation into a verifiable trust anchor. To ensure robust key management, the foundational DICE keys are stored within the RMM rather than in the enclave. This feature prevents insecure key migration and ensures that all cryptographic assets are associated with their respective enclave instance. Because these keys are bound to the hardware, they provide a firm assertion of the platform state. They certify not only the identity of the hosting device but also that the platform booted correctly and is executing the ML workload in a verified, trusted environment. This mechanism allows the system to assert, with hardware guarantees, that the model is running on genuine silicon, effectively mitigating spoofing or emulation attacks.

Ultimately, this combination of memory isolation and hardware-backed identity enables the reliable certification of inference outputs. Once the inference is complete, the Realm releases the ML evaluation output explicitly certified using the private keys derived from the DICE flow. In a typical deployment scenario, a remote relying party (e.g. a cloud auditing service) validates these signed outputs by inspecting an attestation token. This token contains the DICE-derived public key and the measurement log of the Realm. By verifying the token against known golden measurements, the relying party can validate that a genuine version of the ML model, running on authentic hardware, performed the inference.

This architecture offers transformative potential for sectors demanding strict data sovereignty and intellectual property protection. In healthcare, for instance, portable diagnostic devices (e.g., handheld ultrasound scanners) can utilise this hybrid model to process high-resolution imaging data locally. The sensitive patient data remains isolated within the Realm during inference, ensuring compliance with privacy regulations like General Data Protection Regulation (GDPR) [31] or Health Insurance Portability and Accountability Act (HIPAA) [32], even if the device's main operating system is compromised [33]. Similarly, in industrial IoT, equipment manufacturers can deploy proprietary predictive maintenance models to client sites [34]. By sealing the model weights to the device's DICE identity, the manufacturer ensures that their valuable intellectual property cannot be extracted or reverse-engineered by the client, while still providing real-time failure alerts. Finally, in automotive V2X systems, vehicles can broadcast trajectory data signed by the Realm. This allows receiving vehicles to trust that the signal originates from a genuine, uncompromised sensor stack rather than a malicious spoofing device [35].

The security benefits of this architecture are substantial, but they do introduce specific overheads. These costs arise primarily from two sources: the context switching required to move between the Normal World and the Realm, and the additional cycles needed for hardware memory encryption. However, we can mitigate these impacts by strategically placing workloads. We assign heavy, non-sensitive I/O tasks to the Normal World, ensuring that only the core inference logic resides within the Realm. This approach yields substantial gains in both integrity and confidentiality. Ultimately, for high-stakes edge scenarios, these security benefits clearly outweigh the marginal increase in latency.

7. Related Work

Arm CCA represents a nascent technology with emerging hardware support [36, 37, 38, 39]. In contrast, the DICE standard is already well-established within the Confidential Computing landscape. This widespread adoption stems from DICE's architectural flexibility and robust security features. It integrates seamlessly into diverse platforms, ranging from RISC-V to resource-constrained IoT devices [40]. Such versatility validates DICE as a fundamental trust mechanism. Furthermore, this trust is backed by rigorous formal analysis: Tao et al. [41] presented a formally verified implementation of the DICE measured boot. Their work mathematically proved that the protocol is functionally correct and memory-safe. This verification confirms that DICE provides a high-assurance foundation for secure device identities.

For RISC-V, [28] integrates the TCG DICE specification into the Keystone framework. Keystone enables customisable TEEs by relying on a privileged Security Monitor to enforce isolation through Physical Memory Protection primitives. This work rearchitects this monitor to serve as a measured layer within the boot chain. This process roots trust in a hardware Unique Device Secret to drive the derivation of Compound Device Identifiers. By binding these identifiers to the software measurement, the system provisions each enclave with a strong, verifiable identity. The architecture further leverages the DICE chain to generate distinct key pairs for specific scopes. Building on this foundation, recent efforts extend trust also to the network layer [42, 43], ensuring that cryptographic materials and configurations are only provisioned to verified, measured nodes.

On the other hand, [44] presents the first comprehensive formal analysis of the attestation mechanisms in Arm CCA and Intel TDX. The authors develop symbolic models to verify the security properties of the entire attestation stack, from hardware provisioning to the final evidence generation. Their analysis highlights the complexity of the Arm CCA "Composite Attester," where trust is distributed across multiple components rather than a monolithic anchor. By formally specifying these interactions, they identify critical dependencies between the initialisation phases and the validity of the generated evidence. This work highlights the need for a structured approach to gathering and presenting integrity proofs for various components in the ARM CCA architecture.

Recent advancements in Arm CCA have sought to extend hardware-based security beyond standard Realm VMs to achieve finer-grained isolation and higher performance. SHELTER [45] addresses this by utilising the Root world to host a minimal Monitor, securing applications directly in the Normal world without virtualisation overhead. Its multi-GPT design significantly reduces the TCB while establishing an address-space-per-core architecture, a move toward better isolation that aligns well with the principles of integrating DICE for measured execution. Similarly, leveraging the Root world's high privileges, SCRUTINIZER [46] introduces the first secure forensics solution for compromised TrustZone systems. By extending the Root world to protect forensic agents and utilising page-table grafting for efficiency, SCRUTINIZER enables memory acquisition and instruction tracing that are immune to manipulation by privileged adversaries. Together, these works demonstrate a shift toward utilising CCA's RME primitives not just for VM confidentiality, but as a root of trust for granular application isolation and secure system introspection.

8. Discussion and Future Work

This architecture addresses the current Arm CCA ecosystem's lack of hardware-rooted identity derivation. By embedding the DICE layering model into the firmware stack, we enable robust identity provisioning that is cryptographically bound to both platform and software states. Unlike standard CCA, which relies on explicit reporting, this approach enforces active identity through unique key pairs and certificates assigned to each isolated application. These credentials provide implicit proof of integrity, allowing workloads to verify their state and identity to remote peers autonomously. This capability establishes the fundamentals for a comprehensive attestation system within Arm CCA, supporting zero-trust principles.

Building on this foundation, future research will focus on developing secure protocols for authorised Realm migration to facilitate confidential, cloud-native workflows. By leveraging DICE identities for mutual authentication, workloads can verify destination platforms before transferring state, ensuring continuity of trust across the edge-cloud continuum. Additionally, we aim to develop secure update mechanisms for the RMM that accommodate changes in measurement chains. This requires protocols that allow key derivation from valid older RMM versions to preserve data access while strictly enforcing anti-rollback measures to prevent execution on vulnerable firmware. Finally, we envision utilising DICE identities to secure inter-Realm communication via confidential shared memory. This would facilitate TLS-like encrypted channels established directly between Realms, ensuring that the hypervisor remains a blind transport medium. Ultimately, the combined use of CCA isolation and DICE-derived identities offers a scalable, standardised basis for trustworthy multi-tenant computation.

9. Conclusion

This work has presented an architecture that enriches Arm CCA with a DICE-compliant identity and key-derivation hierarchy, enabling Realms to obtain cryptographic identities that are both hardware-rooted and tied to their measured initial state. By mapping the DICE layering model onto the CCA firmware stack and extending the Realm creation with the derivation of $CDI_2(R)$, the design establishes a chain of trust from immutable hardware to the Realm instance. This identity-chain supports explicit attestation through the RAK and introduces the RIK as a DICE-anchored, IEEE 802.1AR-style identity suitable for secure communication. The resulting integration preserves the isolation semantics of RME and the attestation model of the RMM while enabling implicit identity guarantees that CCA alone does not provide.

The case study on confidential ML inference illustrates that these mechanisms are not merely architectural refinements but practical enablers of verifiable, multi-tenant execution on resource-constrained systems. By combining strong hardware isolation with measurement-derived identity, the architecture ensures that confidentiality, integrity, and provenance can be upheld even in adversarial edge environments. More generally, the proposed design facilitates the integration of CCA devices into trust infrastructures that rely on DICE semantics for supply-chain validation, workload integrity, and secure orchestration.

Acknowledgments

This work was partially supported by the SERICS project [PE00000013] under the NRRP MUR program, funded by the European Union-NextGenerationEU. This publication is also part of the project PNRR-NGEU which has received funding from the MUR – DM 352/2022.

Declaration on Generative AI

During the preparation of this work, the authors utilised Gemini and Grammarly to assist with grammar and spelling checks, as well as to paraphrase and reword text for clarity. Following the use of these services, the authors reviewed and edited the resulting content and assume full responsibility for the final publication.

References

[1] L. Ferro, E. Bravi, S. Sisinni, A. Lioy, Privacy Preserving Container Attestation, Journal of Network and Systems Management 34 (2025). doi:10.1145/3586040.

[2] C.-Y. Yang, G. Ramshankar, N. Eliopoulos, P. Jajal, S. Nambiar, E. Miller, X. Zhang, D. J. Tian, S.-H. Chen, C.-F. Perng, Y.-H. Lu, Securing Deep Neural Networks on Edge from Membership

Inference Attacks Using Trusted Execution Environments, in: 29th ACM/IEEE International Symposium on Low Power Electronics and Design, Newport Beach (CA, USA), 2024, pp. 1–6. doi:10.1145/3665314.3670821.

[3] E. Rodríguez, B. Otero, R. Canal, A Survey of Machine and Deep Learning Methods for Privacy Protection in the Internet of Things, Sensors 23 (2023) 1252. doi:10.3390/s23031252.

[4] L. Song, P. Mittal, Systematic Evaluation of Privacy Risks of Machine Learning Models, in: 30th USENIX Security Symposium, online (virtual format), 2021, pp. 2615–2632. https://www.usenix.org/conference/usenixsecurity21/presentation/song.

[5] M. Isakov, V. Gadepally, K. M. Gettings, M. A. Kinsy, Survey of Attacks and Defenses on Edge-Deployed Neural Networks, in: IEEE High Performance Extreme Computing Conference (HPEC), Waltham (MA, USA), 2019, pp. 1–8. doi:10.1109/HPEC.2019.8916519.

[6] F. Fraile, T. Tagawa, R. Poler, A. Ortiz, Trustworthy Industrial IoT Gateways for Interoperability Platforms and Ecosystems, IEEE Internet of Things Journal 5 (2018) 4506–4514. doi:10.1109/JIOT.2018.2832041.

[7] S.-H. Lee, J.-S. Kim, J.-S. Seok, H.-W. Jin, Virtualization of Industrial Real-Time Networks for Containerized Controllers, Sensors 19 (2019) 4405. doi:10.3390/s19204405.

[8] M. Cinque, D. Cotroneo, L. De Simone, S. Rosiello, Virtualizing mixed-criticality systems: A survey on industrial trends and issues, Future Generation Computer Systems 129 (2022) 315–330. doi:10.1016/j.future.2021.12.002.

[9] K. Dolui, C. Kiraly, Towards Multi-Container Deployment on IoT Gateways, in: IEEE Global Communications Conference (GLOBECOM), Abu Dhabi (United Arab Emirates), 2018, pp. 1–7. doi:10.1109/GLOCOM.2018.8647688.

[10] L. Ferro, E. Bravi, S. Sisinni, A. Lioy, SAFEHIVE: Secure Attestation Framework for Embedded and Heterogeneous IoT Devices in Variable Environments, in: 2024 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems, SaT-CPS '24, Association for Computing Machinery, Porto (Portugal), 2024, p. 41–50. doi:10.1145/3643650.3658609.

[11] Arm Limited, ARM Security Technology: Building a Secure System using TrustZone Technology, 2009. https://developer.arm.com/documentation/PRD29-GENC-009492/c.

[12] S. Pinto, N. Santos, Demystifying Arm TrustZone: A Comprehensive Survey, ACM Computing Surveys 51 (2019) 1–36. doi:10.1145/3291047.

[13] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, E. Stapf, SANCTUARY: ARMing TrustZone with User-space Enclaves, in: 26th Annual Network and Distributed System Security Symposium (NDSS), San Diego (CA, USA), 2019. https://www.ndss-symposium.org/ndss-paper/sanctuary-arming-trustzone-with-user-space-enclaves/.

[14] A. Limited, Arm® Realm Management Extension (RME) System Architecture, DEN0129 A.a Non-confidential, 2021. https://documentation-service.arm.com/static/60d3309b677cf7536a55bae0.

[15] Arm Limited, Realm Management Monitor specification, 2024. https://developer.arm.com/-/cdn-downloads/permalink/Architectures/Armv9/DEN0137_1.0-rel0-rc1_rmm-arch_external.pdf.

[16] S. Frost, T. Fossati, G. Mandyam, Arm's Confidential Compute Architecture Reference Attestation Token, IETF Internet-Draft, 2025. https://datatracker.ietf.org/doc/draft-ffm-rats-cca-token/02/.

[17] Trusted Computing Group, Hardware Requirements for a Device Identifier Composition Engine, Version 1.0, Revision 0.91, 2024. https://trustedcomputinggroup.org/resource/hardware-requirements-for-a-device-identifier-composition-engine/.

[18] Trusted Computing Group, DICE Layering Architecture, Version 1.0, Revision 0.19, 2020. https://trustedcomputinggroup.org/resource/dice-layering-architecture/.

[19] L. Ferro, F. Ciravegna, F. Zaritto, A. Lioy, L. Landeiro Ribeiro, et al., Metrics gathering for ai-based trust assessment, in: Proceedings on The 3rd International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS2025), IEEE, September 1-4, 2025.

[20] A. Lioy, G. Ramunno, Trusted Computing, in: P. Stavroulakis, M. Stamp (Eds.), Handbook of Information and Communication Security, Springer, 2010, pp. 697–717. doi:10.1007/978-3-642-04117-4_32.

[21] Confidential Computing Consortium, 2022. https://confidentialcomputing.io/about/.

[22] F. Zaritto, E. Bravi, S. Sisinni, A. Lioy, Extending Kubernetes for Pods Integrity Verification, Journal of Network and Systems Management 34 (2026) 14. doi:10.1007/s10922-025-09988-z.
[23] L. Ferro, A. Lioy, et al., Standard-Based Remote Attestation: The Veraison Project, in: Italian Conference on Cyber Security 2024 (ITASEC24), Salerno (Italy), April 08-12, 2024, pp. 1–13.
[24] Global Platform, TEE System Architecture v1.3, Internet-Draft, 2022. https://globalplatform.org/wp-content/uploads/2022/05/GPD_SPE_009-GPD_TEE_SystemArchitecture_v1.3_PublicRelease_signed.pdf.
[25] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, D. Song, Keystone: an open framework for architecting trusted execution environments, in: A. Bilas, K. Magoutis, E. P. Markatos, D. Kostic, M. I. Seltzer (Eds.), EuroSys '20: Fifteenth EuroSys Conference 2020, ACM, Heraklion (Greece), 2020, pp. 38:1–38:16. URL: https://doi.org/10.1145/3342195.3387532. doi:10.1145/3342195.3387532.
[26] Arm Limited, Confidential Compute Architecture, 2026. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture.
[27] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, I. Verbauwhede, Hardware-Based Trusted Computing Architectures for Isolation and Attestation, IEEE Transactions on Computers 67 (2018) 361–374. doi:10.1109/TC.2017.2647955.
[28] E. Bravi, S. Sisinni, L. Ferro, V. Donnini, A. Lioy, Implementation of the TCG DICE Specification Into the Keystone TEE Framework, IEEE Access 13 (2025) 142284–142303. doi:10.1109/ACCESS.2025.3596829.
[29] IEEE Standards Association, IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity, IEEE Std 802.1AR-2018, 2018. https://standards.ieee.org/standard/802_1AR-2018.html.
[30] Trusted Computing Group, DICE Protection Environment, Version 1.0, 2024. https://trustedcomputinggroup.org/wp-content/uploads/DICE-Protection-Environment-Version-1.0_pub.pdf.
[31] GDPR.EU, What is GDPR, the EU's new data protection law?, 2026. https://gdpr.eu/what-is-gdpr/.
[32] U.S. Department of Health and Human Services, Health Information Privacy, 2026. https://www.hhs.gov/hipaa/index.html.
[33] G. A. Kaissis, M. R. Makowski, D. Ruckert, R. F. Braren, Secure, privacy-preserving and federated machine learning in medical imaging, Nature Machine Intelligence 2 (2020) 305–311. doi:10.1038/s42256-020-0186-1.
[34] P. Li, J. Huang, S. Zhang, C. Qi, Proactive Intellectual Property Protection for Edge AI Models, in: Network and Parallel Computing, Springer Nature Singapore, 2025, pp. 63–74. doi:10.1007/978-981-96-2864-3_6.
[35] B. Groza, P.-S. Murvay, Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks, IEEE Transactions on Information Forensics and Security 14 (2019) 1037–1051. doi:10.1109/TIFS.2018.2869351.
[36] X. Xu, W. Wang, Y. Wu, C. Wang, H. Zhu, H. Ma, Z. Min, Z. Pang, R. Hou, Y. Jin, virtcca: Virtualized arm confidential compute architecture with trustzone, arXiv preprint arXiv:2306.11011 (2023). doi:10.48550/arXiv.2306.11011.
[37] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, G. Stockwell, Design and Verification of the Arm Confidential Compute Architecture, in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), USENIX Association, Carlsbad, CA, 2022, pp. 465–484. URL: https://www.usenix.org/conference/osdi22/presentation/li.
[38] A. C. J. Fox, G. Stockwell, S. Xiong, H. Becker, D. P. Mulligan, G. Petri, N. Chong, A verification methodology for the arm confidential computing architecture: From a secure specification to safe implementations, Proc. ACM Program. Lang. 7 (2023). doi:10.1145/3586040.
[39] C. Wang, K. Lu, F. Zhang, Y. Deng, K. Leach, J. Cao, Z. Ning, S. Yan, T. Wei, Z. He, Building confidential accelerator computing environment for arm cca, IEEE Transactions on Dependable and Secure Computing (2025) 1–18. doi:10.1109/TDSC.2025.3615787.
[40] L. Jäger, R. Petri, Dice harder: a hardware implementation of the device identifier composition engine, in: Proceedings of the 15th International Conference on Availability, Reliability and

Security, ARES '20, Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3407023.3407028.

[41] Z. Tao, A. Rastogi, N. Gupta, K. Vaswani, A. V. Thakur, DICE*: A formally verified implementation of DICE measured boot, in: 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, 2021, pp. 1091–1107. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/tao.

[42] F. Ciravegna, G. Bruno, A. Lioy, IKE-less IPsec: Enhancing Security for IoT Device Networks, in: Italian Conference on Cyber Security 2024 (ITASEC24), Salerno (Italy), April 08-12, 2024, pp. 1–13.

[43] G. D'Onghia, F. Ciravegna, G. Bruno, M. A. Elorza Forcada, A. Pastor, A. Lioy, Securing 5G: Trusted Execution Environments for Centrally Controlled IPsec Integrity, in: 2024 IFIP Networking Conference (IFIP Networking), Thessaloniki (Greece), June 03-06, 2024, pp. 595–597. doi:10.23919/IFIPNetworking62109.2024.10619852.

[44] M. U. Sardar, T. Fossati, S. Frost, S. Xiong, Formal Specification and Verification of Architecturally-Defined Attestation Mechanisms in Arm CCA and Intel TDX, IEEE Access 12 (2024) 361–381. doi:10.1109/ACCESS.2023.3346501.

[45] Y. Zhang, Y. Hu, Z. Ning, F. Zhang, X. Luo, H. Huang, S. Yan, Z. He, Complementing Confidential Computing Environment for Applications on Arm CCA, IEEE Transactions on Dependable and Secure Computing (2025) 1–18. doi:10.1109/TDSC.2025.3604008.

[46] Y. Zhang, F. Zhang, X. Luo, R. Hou, X. Ding, Z. Liang, S. Yan, T. Wei, Z. He, SCRUTINIZER: Towards Secure Forensics on Compromised TrustZone, 2025. doi:10.14722/ndss.2025.230147.