Shared Responsibility in Multi-Tenant Microcontrollers

Bastien Buil,
Chrystel Gaber, Sylvain Plessis
Orange Research
Caen, France
{firstname.lastname}@orange.com

Emmanuel Baccelli
TRiBE
Inria
France
emmanuel.baccelli@inria.fr

Samia Bouzefrane
Cedric
Cnam
Paris, France
samia.bouzefrane@lecnam.net

***Abstract*—Lightweight software containerization solutions execute multiple payloads from several mutually distrusting stakeholders on a resource-constrained microcontroller. This paradigm shifts the accountability model from a single-accountable-actor model where there is only one integrator responsible for the entire monolithic code to a multiple-accountable-actor model where multiple stakeholders share responsibilities. This paper explores this model through three dimensions: responsibility distribution, fulfillment of Cloud commitments, and commitment verification mechanisms along with stakeholders' access to them.**

***Index Terms*—multi-tenancy, responsibility, micro-controllers, eSIM, iSIM, Continuum Cloud**

I. Introduction

With the introduction of lightweight containerization solutions such as WebAssembly Micro Runtime [1] or FemtoContainers [2], the benefits of containerization—enhanced portability, isolation, simplified management—are now accessible for microcontrollers. Thus, commercial solutions for orchestrating containers on resource-constrained IoT devices, such as Atym [3] and MicroEJ [4], have emerged. Furthermore, embedded SIMs and integrated SIMs are also multi-tenant microcontrollers as JavaCard provides a runtime environment that allows the instantiation and management of multiple telecom profiles. Such microcontrollers no longer have monolithic firmwares under the responsibility of a single entity.

This paper aims to analyze and evaluate the shift from a single-actor to a multi-actor accountability model enabled by lightweight containerization on resource-constrained microcontrollers, focusing on responsibility distribution, compliance with cloud commitments, and verification mechanisms.

Related Work: Surveys such as [5], [6] provide a broad picture of IoT general architectures with the common protocols, applications, and components of IoT architecture. Other surveys provide an overview of aspects of microcontroller security like remote attestations [7], [8], secure execution environments using Trusted Execution Environment [9] or virtualization [10].

Accountability has been studied in IoT, however, existing studies always consider the problem of resources and tasks allocation in a network of devices rather than resource allocation to containers on a device. Existing studies concern, for instance, data management [11], IoT decision-making in distributed situation [12], [13], and accountability of AI results in the context of IoT [14]. Quality-of-Service (QoS) at different levels of the IoT architecture has notably been studied by Qu et al. [15] for a network of devices, but they do not analyze QoS management for a single constrained device. The topic of accountability management in multi-tenant infrastructures has already been studied in the context of web services [16] and of Cloud infrastructure [17], [18]. However, to our knowledge, no prior study focuses on accountability management in the context of services from multiple parties hosted on microcontrollers.

Funding: The research leading to these results was funded by ANRT Convention Cifre n°2024/0426.

Contributions: This paper gives a structured overview of the multi-tenant microcontrollers paradigm which shifts the accountability model from a single-actor to a multi-actor framework involving multiple stakeholders. It explores three key dimensions: first, how responsibility is distributed among stakeholders in real-world use cases; second, how existing lightweight containerization solutions address cloud-related commitments to extend the Cloud Continuum into the IoT domain; and finally, the methods for verifying commitments and stakeholders' access to these mechanisms.

Outline: Section II presents multi-tenant containerization and related terminology. Section III analyzes how responsibility is distributed among stakeholders in real-world use cases. Section IV compares existing lightweight containerization solutions in terms of how they address cloud-related commitments. Section V examines the methods used for verifying commitments and the accessibility of these mechanisms to stakeholders. Finally, Section VI presents the next steps for advancing the paradigm of multi-tenant constrained microcontrollers.

II. Multi-tenant containerization on microcontrollers

Containers are lightweight, portable units of software that package an application, its dependencies, and its configuration together, allowing it to run consistently across different computing environments, effectively decoupling application code from the hardware and operating system. Zandberg et al. [2] highlight three categories of use-cases of containerization on low-power IoT: the hosting and isolation of high-level business functions, the hosting and isolation of network debugging and network monitoring code, and the hosting and isolation of

978-3-903176-75-1 ©2025 IFIP

functions managed by multiple tenants. This paper focuses on containerization for multi-tenant applications hosting.

Multi-tenancy refers to a single system utilized by multiple entities, known as tenants. There are two primary types of multi-tenant microcontrollers. The first one is multi-tenant network of nodes in which a node can execute software from multiple entities, but only the applications from one tenant can run at the same time on a device at a given time. This scenario is commonly seen in testbeds and decentralized networks of nodes, such as FIT IoT-LAB[1]. The second type consists of multi-tenant devices, where applications from multiple entities can run simultaneously on one device. The focus of this paper is on the second type. Thus, in this paper, **multi-tenancy** refers to the execution of code from multiple entities on another entity's device, while **multi-tenant microcontrollers** are shared microcontrollers running services from multiple tenants. Tenants deploying their services on a microcontroller are called **Service Providers (SP)**. Typical scenarios we consider incur from the platformization of services deployable on microcontrollers and the emergence of a marketplace.

A key aspect needed for multi-tenancy is ensuring isolation between services from different tenants. On multi-tenant devices, this isolation can be achieved by **software containerization**, which involves running and isolating software units by using lightweight virtualization or hardware isolation. A single isolated software unit with its configuration is referred to as a **container**. A typical multi-tenant constrained microcontroller is represented in figure 1, where multiple service providers are deploying their containers on a microcontroller owned by another entity, the **Microcontroller Provider (MP)**. The MP buy the device to a **Device Manufacturer (DM)**. The microcontroller and a remote management platform are managed by the **Container Management Operator (CMO)**, which operates the deployment and operation of containers.

Containerization can be classified into two types of isolation guarantees. The first type involves preventing containers from accessing system resources or resources of other containers. The second type focuses on restricting applications of the system and containers to have access to other containers.

When deploying containers on a device, tenants might want guarantees that their containers will have the appropriate resources to work. One type of guarantee is contracts between the maintainer of the device and tenants defining the commitments on the provided hosting service. These contracts, named **Service Level Agreements (SLA)**, contain metrics and target values on the service that will be provided. SPs might not entirely trust MPs and CMOs to respect committed SLA or cannot always do a legal contract with CMOs. Thus, Service Providers might want to verify by themselves the provided service-level by using mechanisms, that we call **SLA verification methods**, that are mechanisms used to verify that the SLAs are respected.

III. Responsibility Distribution

Similarly to the shared responsibility model for Cloud [19], we define a responsibility model which defines how responsibilities are distributed among stakeholders involved in the multi-tenant microcontroller architecture. Table I describes a general model of responsibilities and its declination in three use cases of multi-tenant microcontrollers.

[1]https://www.iot-lab.info

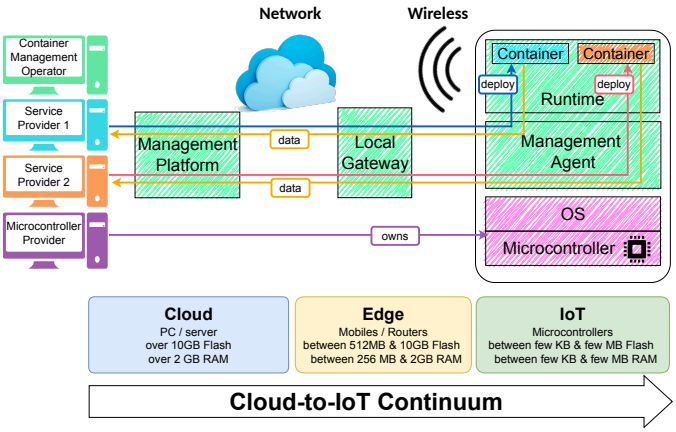

Fig. 1. Cloud-to-IoT Continuum and the multi-tenant microcontrollers paradigm

A. Use cases description

The first use case "IoT-based servitization" is the process where companies enhance their IoT devices with additional services which can be provided by third parties. An example of this use case, presented in [20], is a smart meter manufacturer allowing utility companies to customize their product with their own services thanks to a Container Management Infrastructure proposed by the commercial solution [4]. Through the solution provided by Container Management Operator (CMO), smart meters companies become IoT Providers (IoT-P) and utility companies become Service Providers (SP) for the smart meter.

The second and third use cases are related to embedded SIMs (eSIM) [21] and integrate SIMs (iSIM) which are relatively close. The main difference between both is that the eSIM is an independent Secure Element provided by eSIM manufacturers and soldered to the Device during the integration of its hardware components while the iSIM is built into a dedicated Tamper Resistant Element (TRE) inside the device's chipset. Similarly to SP, Communication Service Providers (CSP) deploy their profiles on eSIMs/iSIMs through a Subscription Manager and Data Preparation+ (SM-DP+) platform.

B. Responsibility model

In the context of multi-tenant microcontroller architectures, we define a responsibility model comprising seven key responsibility areas. In this section, we use these areas as criteria to identify stakeholder roles and responsibilities within the use cases presented above.

1) Physical Device: This responsibility consists in assembling the physical device by integrating hardware components. Typically, this role is held by the Device Manufacturer. In IoT-based Servitization, this role is held by IoT Providers (IoT-P) such as the smart meter manufacturer who creates IoT devices before opening them to third-party applications. In the eSIM

TABLE I
RESPONSIBILITY MODEL OF MULTI-TENANT MICROCONTROLLERS

Responsibility	General model	Use Cases		
		IoT-based Servitization	eSIM	iSIM
Application (accounts, data, code)	SP	SP	CSP	CSP
Container packaging	SP $\vee$ CMO	SP $\vee$ CMO	SM-DP+ $\wedge$ CSP	SM-DP+ $\wedge$ CSP
Container deployment	CMO	CMO	SM-DP+	SM-DP+
Container Management Infrastructure	CMO	CMO	SM-DP+ $\wedge$ OEM	SM-DP+ $\wedge$ OEM
Microcontroller Firmware	MP	IoT-P $\wedge$ CMO	eSIM-M	CPU-M
Network	DM	IoT-P	OEM	OEM
Physical Device	DM	IoT-P	OEM	OEM

Roles: DM: Device Manufacturer, MP: Microcontroller Provider, CMO: Container Management Operator, SP: Service Provider
Actors: CMO: Container Management Operator, CPU-M: CPU Manufacturer, CSP: Communications Service Provider, eSIM-M: eSIM Manufacturer, IoT-P: IoT Provider, OEM: Original Equiment Manufacturer, SM-DP+: Subscription Manager and Data Preparation+, SP: Service Provider

and iSIM ecosystem, this responsibility is held by Original Equipment Manufacturers (OEMs) who build mobile phones and IoT devices where an eSIM or an iSIM can be used.

2) Network: This responsibility involves ensuring connectivity and network-related functionalities within the device so that it can interact with the infrastructure where it is ultimately deployed. This responsibility is primarily managed by the Device Manufacturer. In the case of the IoT-based Servitization, IoT-P ensures that the device contains a component for network connectivity. IoT-P may also set up the network infrastructure to create a fleet of devices that can communicate with IoT-P servers. In the case of eSIM and iSIM, the OEM manages this responsibility by integrating a modem which interacts with the eSIM or iSIM to provide connectivity.

3) Microcontroller Firmware: This responsibility corresponds to the creation, configuration, and management of the firmware running on the microcontroller. Typically, this responsibility is held by the Microcontroller Provider (MP). An example of Microcontroller Provider is the eSIM manufacturer in eSIM ecosystems or the chipset manufacturer in iSIM ecosystems. In IoT-based servitization use cases, this responsibility is shared between IoT-P and the Container Management Operator (CMO) which provides an agent that must be installed within the firmware by IoT-P.

4) Container Management Infrastructure: This responsibility involves operating and maintaining the platform and agents responsible for managing containers within the device. Usually, this role is assigned to a CMO. In the context of eSIM and iSIM ecosystems, the Subscription Manager and Data Preparation+ (SM-DP+) provides the infrastructure to preparation profiles and store them securely before deploying them remotely on eSIMs or iSIMs. However, this infrastructure also relies on the presence of a Local Profile Assistant or an equivalent software interface deployed in the Physical Device to streamline downloading, installing, and managing telecommunication profiles. Therefore, the responsibility is shared between the SM-DP+ and the OEM.

5) Container Deployment: This responsibility consists in deploying containers on one or multiple devices, which is performed by the Container Management Infrastructure following the instructions of the SP. Thus, this task falls under the CMO's responsibility. Our observations across all three use cases confirm this, as the CMO and the SM-DP+ are responsible for deploying containers on IoT devices or managing telecom profiles on eSIMs and iSIMs.

6) Container Packaging: This responsibility corresponds to packaging the application as one or multiple containers which can be done either by the CMO or the SP. In IoT-based servitization, this task is carried out either by the SP or the CMO depending on the CMO's offer. The SP can package its application itself using the CMO's infrastructure or can delegate it to the CMO. In the eSIM and iSIM use case, the preparation of the telecom profile is performed by the SM-DP+.

7) Application: The last responsibility area consists in developping, maintaining applications to provide the Service operated by the SPs, such as the utility companies in the smart meter servitization scenario or telecommunications providers in eSIM and iSIM ecosystems.

C. Discussion

This section analyzes the responsibility model in multi-tenant microcontroller architectures, focusing on seven key responsibility areas.

Our analysis across three distinct use cases—IoT-based servitization, eSIM, and iSIM ecosystems—shows that responsibilities in the multi-tenant microcontroller paradigm depend on context but follow consistent patterns across diverse use cases. For instance, the physical device assembly and network integration are primarily managed by DM or OEMs, depending on the ecosystem, while firmware management and container orchestration responsibilities are shared or delegated to specialized entities like the CMO and SM-DP+.

Across all use cases, the task of deploying containers—whether applications, profiles, or telecommunication configurations—is consistently performed by the CMO or equivalent entities following instructions from the SP.

The responsibility for packaging applications as containers varies depending on the ecosystem. In IoT-based servitization, this task can be delegated to either the SP or the CMO, reflecting different operational models and levels of control. In contrast, in eSIM and iSIM ecosystems, profile preparation and packaging are typically handled by the SM-DP+.

With responsibilities being delegated to CMOs, MPs, and DMs, SPs depend on other parties to fulfill their commitments to their customers, posing a responsibility risk. As stakeholders can distrust each other, there is the risk of dishonest or honest but greedy stakeholders. To mitigate this risk and clarify the responsibility in the event of an incident, stakeholders can designate an arbitrator to monitor commitments or each stakeholder can individually monitor commitments own to it.

IV. Commitments

In the context of the Cloud Continuum, one might expect the commitments defined in the Cloud to also apply to multi-tenant microcontrollers. This section's objective is to analyze whether containerization solutions can cover such types of commitments. To achieve this, we first identify applicable commitments applicable in Cloud ecosystem. Then, we analyze whether existing containerization solutions for the three previously defined use cases can cover the identified commitments. Finally, we discuss on the need for more guarantees in constrained microcontrollers solutions to allow Cloud-like commitments.

A. Overview of commitments in Cloud infrastructure

Based on the commitments defined in SLALOM D3.2 [27] and Cloud Service Level Agreement Standardization Guidelines [28], we identify four key categories of Cloud commitments: performance, security, monitoring, and governance.

a) Performance commitments: Cloud hosts typically commit on performance and resources, this is mainly done by giving exclusive access to physical and logical resources. The main performance-related commitments are related to availability, elasticity, network, hardware resources, application capacity, and energy consumption. **Availability** is often represented by the percentage of time the device should be available to be used, this is linked with functioning of the device, but this often takes into account the connectivity of the device to the internet. **Elasticity** corresponds to the ability of a service to quickly provision or deprovision resources for a service. **Network** commitments are often on one metric as response time, throughput, and bandwidth. For hardware resources, Cloud hosts typically commit on available hardware like the type of **CPU**, its percentage of time dedicated to the application, allocated size of **memory**, size and type of **storage**, access to **peripherals**. While commitments mostly concerns hardware resource and dedicated execution time, for SaaS (Software as a Service), the host can commit on **application-related** commitments like the number of concurrent connections.

b) Security commitments: Security commitments are generally taken on **backup** of deployed applications, access control used for the management of the service, and **authentication methods** to access this service. Cloud hosts can also commit on the reinforced **isolation** of its service, and can propose dedicated hardware such as storage, peripherals, or dedicated machines to enhance isolation, and proposed **secure network communications** and secure storage usable by applications.

c) Monitoring commitments: Another type of commitment concerns monitoring and certification. Cloud hosts can provide monitoring services for deployed applications. **Application monitoring** often consists of monitoring the functioning and performance of deployed applications, while **security monitoring** consists of monitoring the security and integrity of the applications. Cloud can monitor security by monitoring the network traffic with Intrusion Detection Systems (IDS) or by auditing of the running OS and application.

d) Governance & Policy commitments: The last type of existing Cloud commitments concern the administrative domain. This type of commitment ranges from contract-related terms as reversibility and termination, to data management with data life cycle, portability, and usage. Another example of administrative commitment is the Cloud Provider's support policy, describing what kind of help its support team is expected to perform. In the rest of the document, this commitment is not considered as it does not relate with technical measures.

B. Commitments coverage by containerization solutions

This section evaluates the guarantees provided by containerization solutions for microcontrollers with regard to the commitments defined in the previous section. While all solutions we have evaluated are runtime environments, Atym [3], MicroEJ [4] and Toit [25] also propose a platform to deploy and manage containers on their target. Our analysis is summarized in Table II. The governance commitments are not examined since such commitments are covered by administrative actions and not by technical capacities of containerization solutions.

1) Partial coverage of commitments on performance: Containerization solutions use techniques to prevent a container from monopolizing the device's resources. This can be achieved by reserving, either fully or partially, access to specific system resources for a container. Containerization systems can grant containers granular access to system resources, such as limiting amounts of memory and storage per container [22], [23], providing internet access with data quotas [23], limiting access to a finite number of peripherals [22], [23], and constraining the electrical consumption of containers [22]. To prevent a container from monopolizing execution time, container orchestration mechanisms are necessary. Such orchestration can be achieved by delegating container management to the operating system [2], by implementing priority-based preemptive schedulers for containers [23], or by implementing a preemptive scheduler allocating a percentage of CPU time to containers [24].

2) Thorough Security commitments: Containerization does by definition memory isolation between containers to have spatial isolation between containers, so all studied solutions provide it. Containerization solutions can also include security mechanisms like authentication for management and deployment of containers [3], [4], [25], over-the-air (OTA) update [3], [4], [23], [25], [26], reproducible environments to restore

TABLE II
COMMITMENTS GUARANTEED BY CONTAINERIZATION SOLUTIONS FOR MICROCONTROLLERS

			Performances									Security				Monitoring	
		Use Case	Avail	Elas	Net	CPU	Mem	Stor	Perip	App	Ener	Back	Auth	Iso	Sec Com	App Mon	Sec Mon
Aerogel	[22]	IoTServ	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗
Polyglot Cerberos	[23]	IoTServ	✗	✗	~	~	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
VeloxVM	[24]	IoTServ	✗	✗	✓	✓	✓	~	✓	✗	✓	✗	✗	✓	✓	✗	✗
Toit	[25]	IoTServ	✗	✗	✗	✗	✗	✗	✗	✗	✗	~	✓	✓	✓	~	✗
Atym	[3]	IoTServ	✗	✗	✗	~	✗	✗	✓	✗	✗	~	✓	✓	✓	✓	✗
MicroEJ	[4]	IoTServ	✗	✗	~	✗	✗	✗	✓	✗	✗	✗	✓	✓	✓	~	✗
JavaCard	[26]	eSIM & iSIM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗

Commitments on: **Iso Type**: isolation type, **Iso Tech**: isolation technique, **Avail**: availability, **Elas**: elasticity, **Net**: network, **CPU**: execution time, **Mem**: volatile memory, **Stor**: storage, **Perip**: peripheral access, **App**: application-related commitments, **Ener**: energy consumption, **Back**: service backup and restoration, **Auth**: authentication for device management, **Iso**: isolation, **Sec Com**: secure communication, **App Mon**: application monitoring, **Sec Mon**: security monitoring.
Symbols: ✓: covered by the solution, ✗: not covered by the solution, ~: partially covered by the solution.

containers on devices [3], [25], and secure inter-container and network communication [3], [23], [26].

3) Scarce monitoring commitments: While most runtimes lack integrated monitoring solution, commercial solutions coming with a management platform integrate some monitoring. For example, these solutions can report the running version of the deployed applications [3], [25], the status of the application [3], and custom application-defined metrics [3], [4]. Studied solutions do not integrate security monitoring mechanisms. In current containerization solutions on microcontrollers, monitoring is limited both for security monitoring notably lacking remote attestation systems, and performance-related monitoring lacking reporting precise data on application performances.

C. Discussion

Security seems to be the most researched guarantee of containerization solutions, with for instance numerous research papers on isolation, authentication, and secure remote updates. This maturity of the security of containerization solutions is probably the consequence of the security needs of single-tenant microcontrollers which have faced numerous attacks due to the lack of secure authentication for accessing the device, and vulnerable code that could not be remotely patch.

On the other hand, performance guarantees are only partly supported with guarantees on memory, storage, peripheral access, execution time and network bandwidth, but with no proposition integrating all these guarantees. This is most likely due to the newness of multi-tenant microcontrollers. With single-tenant device, a developer can rely on performance guarantees of devices defined in the datasheet of the device, but in the context of multi-tenancy, this is no longer possible as resources are shared between containers.

Similarly to performance and security commitments, monitoring mechanisms for multi-tenant microcontrollers are limited. For security monitoring, this is probably due that existing containerization solutions, do not yet integrate remote attestation mechanism. For security monitoring, this is probably due to the fact that with single-tenant microcontrollers, the developers do not fear performance irregularities as they are the single tenant of the device, whereas with multi-tenancy, the developer's app might not work properly due to apps of other tenants.

This lack of guarantee results from the low maturity of containerization on IoT that does not yet match cloud commitments. Ensuring service-centric commitments on constrained devices is an open research question, which needs to be researched to have Cloud-like multi-tenant microcontrollers.

V. COMMITMENT VERIFICATION MEANS

This section focuses on the usability of five commitments verification methods by stakeholders in the use cases. We focus in particular on the capacity to verify the commitments of the microcontroller. Table III summarizes the results.

1) Application-level tracing: This method consists in using data collected by the application to infer information related to the microcontroller status. For example, in the IoT-based servitization use case an alert may be raised because the smart meter sends to SP an out-of-range value. CSPs in eSIM and iSIM ecosystems can include in their profile an application to trace events that can be collected later on for analysis. Other actors usually do not collect data at application-level.

2) Microcontroller-level tracing: This method involves analyzing data reported by the microcontroller firmware or the Container Management Agent on the device to assess system health. For example, it is possible to monitor execution flow by instrumenting the code and reporting traces of the execution [29]. Any entity sending data to a backend—such as the IoT-P, CMO, or SP—can use this approach to monitor devices or containers provided that they have the permission to access the traces, which may not be the case of CMO and SP. In the eSIM and iSIM ecosystems, there is no requirement from standards to collect or expose such data.

3) Remote probing: Remote probing consists in probing an exposed endpoint to check its availability and latency. Service-

TABLE III
COMMITMENT VERIFICATION METHODS

	Use cases		
Stakeholder	IoT-based Servitization	eSIM	iSIM
Application-level tracing	SP	CSP	CSP
Microcontroller-level tracing	IOT-P, CMO*	-	-
Remote Probing	IoT-P, CMO*, SP*	SM-DP+	SM-DP+
Monitoring Agent	IoT-P, CMO*	CSP*	CSP*
Remote Attestation	IoT-P, CMO*, SP*	-	-
Network Monitoring	IoT-P, CMO*	OEM, CSP	OEM, CSP

Actors: CMO: Container Management Operator, CPU-M: CPU Manufacturer, CSP: Communications Service Provider, eSIM-M: eSIM Manufacturer, IoT-P: IoT Provider, OEM: Original Equiment Manufacturer, SM-DP+: Subscription Manager and Data Preparation+, SP: Service Provider
*: denotes a limited capacity of action

Level Agreements frameworks like WSLA [30] and ML-SLA-IoT [31] include monitoring components for this purpose.

In the IoT-based servitization use case, this approach requires the device to expose a service on an exposed port, which can be managed by the CMO or the IoT-P via an on-device agent. SP could also be granted access to open endpoints accessible from the network. In the eSIM and iSIM ecosystems, remote probing can theoretically be performed during the provisioning phase by the SM-DP+ as per SGP22 [32].

4) Monitoring agent: This method requires an analyzing data reported by the microcontroller firmware or the Container Management Agent on the device to assess system health. An on-device agent monitors SLA compliance and reports status to a server. Mazhelis et al. [33] propose agents that track information such as battery, memory, and sensor data.

These agents can be used by the CMO or IOT-P in the IoT-based servitization use case. SPs can embed one in its container, though container access may be limited. To our knowledge, such approach generally does not exist in eSIM and iSIM ecosystems. CSPs may monitor data in an applet deployed in their profiles provided that they are exposed by the eSIM-P or CPU-M. The availability of such data is not standardized and therefore is implementation-dependant.

5) Remote Attestations: Remote attestation allows to remotely attest the state of the deployed service and have a shareable proof based on a Trusted Computing Base (TCB). The attestation ususally consists in verifying the memory integrity or control-flow integrity. The memory integrity is used to verify the integrity of executed services, while the control-flow integrity is used to verify the proper execution of services. While remote attestation can be set up by the IoT-P, the CMO and the SP, usually, only the IoT-P has access to a TCB and data allowing to perform these verifications.

6) Network Monitoring: It is possible to verify compliance to commitments by monitoring the network communication of the devices and analyzing the network traffic. Network monitoring can be done directly on the device by using a hardware component which analyzes the network traffic [34], through a middleware, or through an external component such as a hub or a gateway which intercepts and analyzes the network traffic. Hardware components can be set up either by the IoT-P or the OEM while software components can be set up by the CMO. CSPs can use their communications infrastructure (e.g. antennas) to analyse the traffic and behavior from eSIMs or iSIMs.

Our analysis shows a clear disparity between the access of stakeholders to commitments verification data. This is due to the scope of the responsibility each actor has. A particularity is the eSIM and iSIM ecosystem where SM-DP+ and CSPs have very limited access to means to verify commitments of the microcontrollers used. This is due to the fact that the security model in these ecosystems did not evolve much from the SIM security model which heavily relies on security-by-design and where CSPs are SIM providers and therefore hold the responsibility related to the Container Management Infrastructure and Microcontroller Firmware.

We identify two avenues of research to improve SLA verification for multi-tenant microcontrollers. The first one is to adapt existing techniques for Cloud SLA verification, as techniques using side-channels to detect unauthorized virtual machines [35] and shared storage device [36], or techniques using cooperation between stakeholders to verify of the integrity of stored data [37]. The second avenue is to take inspiration from fingerprinting and anomaly detection techniques on IoT to monitor the execution of containers and detect irregularity and variations in the behavior of the device to detect diversion from the negotiated service-level.

VI. CONCLUSION & FUTURE WORKS

This paper explores the paradigm shift from a single-actor to a multi-actor accountability model enabled by lightweight containerization for microcontrollers. By analyzing responsibility distribution, commitment fulfillment, and usability of verification mechanisms for stakeholders, this paper highlights some limitation for responsibility and accountability management in the multi-tenant microcontrollers paradigm.

We show that while existing containerization solutions for microcontrollers provide many security guarantees, they currently offer limited support for performance and monitoring

guarantees comparing to cloud environments. Furthermore, the evaluation of commitment verification methods highlights the need for enhanced mechanisms tailored to constrained devices, especially for Service Providers who heavily rely on Microcontroller Providers and Containerization Management Operators. While responsibility sharing creates a responsibility risk for Service Providers towards its customers, this risk is amplified by the limitations of containerization techniques that cannot guarantee Cloud-like commitments and the lacks of verification methods enabling service providers to monitor the provided service-level.

Looking ahead, future research should focus on designing comprehensive and interoperable SLA frameworks that incorporate verification and enforcement mechanisms suitable for resource-limited environments. Standardization efforts are crucial to facilitate seamless integration, interoperability, and trust among diverse stakeholders. Advancing these areas will help bridging the gap between IoT and cloud paradigms.

REFERENCES

[1] "Bytecodealliance/wasm-micro-runtime," Bytecode Alliance. [Online]. Available: https://github.com/bytecodealliance/wasm-micro-runtime

[2] K. Zandberg, E. Baccelli, S. Yuan, F. Besson, and J.-P. Talpin, "Femto-Containers: Lightweight Virtualization and Fault Isolation For Small Software Functions on Low-Power IoT Microcontrollers," Oct. 2022.

[3] Atym, "Atym products page." [Online]. Available: https://www.atym.io/products

[4] MicroEJ, "MICROEJ VEE Software Container for Embedded Systems." [Online]. Available: https://www.microej.com/product/vee-software-container-embedded-systems/

[5] W. Kassab and K. A. Darabkh, "A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations," *Journal of Network and Computer Applications*, vol. 163, p. 102663, Aug. 2020.

[6] E. Schiller, A. Aidoo, J. Fuhrer, J. Stahl, M. Ziörjen, and B. Stiller, "Landscape of IoT security," *Computer Science Review*, vol. 44, p. 100467, May 2022.

[7] W. A. Johnson, S. Ghafoor, and S. Prowell, "A Taxonomy and Review of Remote Attestation Schemes in Embedded Systems," *IEEE Access*, vol. 9, pp. 142 390–142 410, 2021.

[8] B. Kuang, A. Fu, W. Susilo, S. Yu, and Y. Gao, "A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects," *Computers & Security*, vol. 112, p. 102498, Jan. 2022.

[9] A. Muñoz, R. Ríos, R. Román, and J. López, "A survey on the (in)security of trusted execution environments," *Computers & Security*, vol. 129, p. 103180, Jun. 2023.

[10] K. Grunert, "Overview of JavaScript Engines for Resource-Constrained Microcontrollers," in *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*. Split, Croatia: IEEE, Sep. 2020, pp. 1–7.

[11] A. Crabtree, T. Lodge, J. Colley, C. Greenhalgh, K. Glover, H. Haddadi, Y. Amar, R. Mortier, Q. Li, J. Moore, L. Wang, P. Yadav, J. Zhao, A. Brown, L. Urquhart, and D. McAuley, "Building accountability into the Internet of Things: The IoT Databox model," *Journal of Reliable Intelligent Environments*, vol. 4, no. 1, pp. 39–55, Apr. 2018.

[12] J. Singh, C. Millard, C. Reed, J. Cobbe, and J. Crowcroft, "Accountability in the IoT: Systems, Law, and Ways Forward," *Computer*, vol. 51, no. 7, pp. 54–65, Jul. 2018.

[13] C. Norval, J. Cobbe, and J. Singh, "Towards an accountable Internet of Things: A call for reviewability," Jun. 2021.

[14] M. Humayun, N. Tariq, M. Alfayad, M. Zakwan, G. Alwakid, and M. Assiri, "Securing the Internet of Things in Artificial Intelligence Era: A Comprehensive Survey," *IEEE Access*, vol. 12, pp. 25 469–25 490, 2024.

[15] Z. Qu, Y. Wang, L. Sun, D. Peng, and Z. Li, "Study QoS Optimization and Energy Saving Techniques in Cloud, Fog, Edge, and IoT," *Complexity*, vol. 2020, no. 1, p. 1, 2020.

[16] M. H. Hasan, J. Jaafar, and M. F. Hassan, "Monitoring web services' quality of service: A literature review," *Artificial Intelligence Review*, vol. 42, no. 4, pp. 835–850, Dec. 2014.

[17] J. Yao, S. Chen, and D. Levy, "Accountability for Service Compliance: A Survey," *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*, vol. 3, no. 1, pp. 16–43, 2012.

[18] N. Hamdi, C. El Hog, R. Ben Djemaa, and L. Sliman, "A Survey on SLA Management Using Blockchain Based Smart Contracts," in *Intelligent Systems Design and Applications*, A. Abraham, N. Gandhi, T. Hanne, T.-P. Hong, T. Nogueira Rios, and W. Ding, Eds. Cham: Springer International Publishing, 2022, vol. 418, pp. 1425–1433.

[19] msmbaldwin, "Shared responsibility in the cloud - Microsoft Azure." [Online]. Available: https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility

[20] IoT For All, "Software-Defined IoT and Sustainable Design | MicroEJ's Semir Haddad," Oct. 2023. [Online]. Available: https://www.youtube.com/watch?v=LDOvZ-wctPo

[21] ENISA, "Embedded SIM ecosystem, security risks and measures," ENISA, Tech. Rep., Mar. 2023. [Online]. Available: https://www.enisa.europa.eu/sites/default/files/publications/Embedded%20Sim%20Ecosystem%20Security%20Risks%20and%20Measures.pdf

[22] R. Liu, L. Garcia, and M. Srivastava, "Aerogel: Lightweight Access Control Framework for WebAssembly-Based Bare-Metal IoT Devices," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, Dec. 2021, pp. 94–105.

[23] S. Akkermans, B. Crispo, W. Joosen, and D. Hughes, "Polyglot CerberOS: Resource Security, Interoperability and Multi-Tenancy for IoT Services on a Multilingual Platform," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. New York NY USA: ACM, Nov. 2018, pp. 59–68.

[24] N. Tsiftes and T. Voigt, "Velox VM: A safe execution environment for resource-constrained IoT applications," *Journal of Network and Computer Applications*, vol. 118, pp. 61–73, Sep. 2018.

[25] "Toit - high-level software platform for the ESP32," https://toit.io/.

[26] "Java Card technology," https://www.oracle.com/java/java-card/.

[27] "SLALOM - SLA specification and reference model - D3.2," SLALOM Project, Tech. Rep. D3.2, Oct. 2015.

[28] "Cloud Service Level Agreement Standardisation Guidelines - C-SIG SLA," Cloud Select Industry Group Subgroup on Service Level Agreement (C-SIG-SLA), Tech. Rep., Jun. 2014.

[29] S. Fischmeister and P. Lam, "Time-Aware Instrumentation of Embedded Software," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 652–663, Nov. 2010.

[30] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, no. 1, pp. 57–81, Mar. 2003.

[31] S. Noureddine and B. Meriem, "ML-SLA-IoT: An SLA Specification and Monitoring Framework for IoT applications," in *2021 International Conference on Information Systems and Advanced Technologies (ICISAT)*, Dec. 2021, pp. 1–12.

[32] GSMA, "SGP.22 V3.1." [Online]. Available: https://www.gsma.com/solutions-and-impact/technologies/esim/gsma_resources/sgp-22-v3-1/

[33] O. Mazhelis, M. Waldburger, G. S. Machado, B. Stiller, and P. Tyrväinen, "Retrieving Monitoring and Accounting Information from Constrained Devices in Internet-of-Things Applications," in *7th International Conference on Autonomous Infrastructure (AIMS)*, vol. LNCS-7943. Springer, Jun. 2013, p. 136.

[34] F. Hategekimana, T. J. Whitaker, M. J. Hossain Pantho, and C. Bobda, "IoT Device security through dynamic hardware isolation with cloud-Based update," *Journal of Systems Architecture*, vol. 109, p. 101827, Oct. 2020.

[35] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis," in *2011 IEEE Symposium on Security and Privacy*, May 2011, pp. 313–328.

[36] Z. Wang, K. Sun, S. Jajodia, and J. Jing, "Proof of Isolation for Cloud Storage," in *Secure Cloud Computing*, S. Jajodia, K. Kant, P. Samarati, A. Singhal, V. Swarup, and C. Wang, Eds. New York, NY: Springer, 2014, pp. 95–121.

[37] A. T. Wonjiga, S. Peisert, L. Rilling, and C. Morin, "Blockchain as a Trusted Component in Cloud SLA Verification," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, ser. UCC '19 Companion. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 93–100.