10

Methods for Requirements Engineering, Verification, Security, Safety, and Robustness in AIoT Systems

Marcelo Pasin[1], Jämes Ménétrey[1], Pascal Felber[1], Valerio Schiavoni[1], Hans-Martin Heyn[2], Eric Knauss[2], Anum Khurshid[3], and Shahid Raza[3]

[1]University of Neuchâtel, Switzerland
[2]Gothenburg University, Sweden
[3]Research Institutes of Sweden AB, Sweden
E-mail: marcelo.pasin@unine.ch; james.menetrey@unine.ch; pascal.felber@unine.ch; valerio.schiavoni@unine.ch; hans-martin.heyn@gu.se; eric.knauss@cse.gu.se; anum.khurshid@ri.se; shahid.raza@ri.se

Abstract

This chapter presents methods for requirements engineering, verification, security, safety, and robustness with a special focus on AIoT systems. It covers an architectural framework dealing with requirements engineering aspects of distributed AIoT systems, covering several clusters of concern dealing with the context description of the system, learning environment of the deep-learning components, communication concerns, and a set of quality concerns, such as ethical aspects, safety, power, security, and privacy aspects. Each cluster contains a set of architectural views sorted into different levels of abstraction. In addition, it introduces WebAssembly as an interoperable environment that would run seamlessly across hardware devices and software stacks while achieving good performance and a high level of security as a critical requirement when processing data off-premises. To address security aspects in AIoT systems, remote attestation and certification mechanisms are

introduced to provide a TOCTOU (time-of-check to time-of-use) secure way of ensuring the system's integrity.

Keywords: IoT, machine learning, AIoT, requirements engineering, TOCTOU, WebAssembly, verification, security, safety, robustness.

10.1 Introduction

More and more traditional algorithms are replaced by models based on deep learning. Deep learning has proven to be successful in solving problems of large complexity, such as natural language processing or facial recognition tasks. In addition, systems tend to be broken down into different components, to be placed where they are most needed and can be most efficient. By establishing high-bandwidth connections between all kinds of different devices and allowing many different system configurations, the components of the distributed system become part of what is known as the Internet of Things (IoT). When combining deep learning with the properties of IoT, new concerns might arise that are not yet foreseen by standards and literature. The new concerns include aspects such as data quality, heuristic deep-learning modeling, learning of the models, or even new ethical considerations.

Applying disruptive systems and methods in real-world applications relies on advances in development methodology. New methods for effectively describing requirements for AI-based algorithms that are distributed over IoT devices from edge to the cloud and how they relate to end-user concerns and needs are a crucial part of the solution. These methods build the foundation for specifying components of such systems in a way that enables to reason about robustness and safety as well as to enable security, privacy, and trust by design. AIoT systems contain both traditional software and hardware components and AI components running on specialized AI acceleration hardware. The challenge is not only to specify and design the AI components but also to integrate them together with the traditional components into an overall AI-enabled system.

10.2 Architecture Framework for AIoT Systems

Architecture frameworks (AF) provide a reusable knowledge structure for designing an AIoT system. An AF organizes architectural descriptions into different architectural views [6]. Different architectural views allow for decomposing the design task into smaller and specialized subtasks, each task specifically suitable to serve a certain design aspect of the system.

10.2.1 State-of-the-art for AI systems architecture

In a research agenda for engineering AI systems, the authors provide a list of challenges when developing architectures for systems with AI components [7]: Providing the right (quality of) data used for training, establishing the right learning infrastructure, building a sufficient storage and computing infrastructure and creating a suitable deployment infrastructure. The latter includes monitoring the behavior of the AI systems under operation because it might only be possible to detect and correct flaws in an AI system after deployment. Furthermore, AI systems do not only consist of AI components but also rely also on conventional software and hardware components. The development of AI components and traditional system components must therefore be aligned to avoid unwanted technical debt [11]. However, as Woods emphasizes, traditional architecture frameworks, such as the 4+1 architectural view model by Kruchten [9], do not account for data and algorithm concerns connected to AI component development [10]. Generally, new stakeholders (e.g., data engineers, or governmental agencies overseeing the use of AI in society), and new concerns connected to AI like data quality aspects, ethical considerations such as fairness or explainability, and eventually many more, need to be represented through new architectural viewpoints. An example of such an additional viewpoint is a learning viewpoint governing the view on the machine learning flow [12]. Developing AI components is a hierarchical, yet also iterative task: Prepare training data and environment, create a suitable model, train and evaluate the model, tune, and repeat training, and eventually deploy and monitor the runtime behavior of the trained model [7, 13]. To fulfill a stakeholder's goal with a system, its design needs to be decomposed into different levels of system design, and consistency needs to be ensured to satisfy high-level requirements [14]. In addition, the system design must also allow for **middle-out development**, where existing components need to be integrated into the overall system design (e.g., transfer-learning from existing AI models or integration of off-the-shelf components). Murugesan et al. propose a hierarchical reference model which supports the appropriate decomposition of requirements to the composition of the system's components [15]. In their model, they define how components can be decomposed into subcomponents. To ensure consistency between the system architecture and the requirements, they define the terms consistency, satisfaction, and acceptability. One major advantage of their model is that, if the decomposition of system components is done correctly, these components can be independently specified and developed.

In summary, a major challenge in AI system design is the lack of design patterns, standards, and reference architectures that support the co-design of traditional software components and AI components [16]. When designing a system, a range of quality aspects, such as safety, security, and privacy needs to be considered. For AI systems, ethical aspects such as explainability of decisions, fairness, and participation play an important role during the system design process. Therefore, the architectural framework for AIoT shall not only support the seamless design and integration of traditional software components and AI components but also allow for all necessary quality concerns to be considered as early as possible in the design process.

10.2.2 A compositional architecture framework for AIoT

The main goal is to introduce an architecture framework based on compositional thinking suitable for developing distributed AI-based systems. The idea of an architectural framework is to provide a knowledge structure that allows the division of an architectural description into different architectural views [6]. An architectural view expresses "the architecture of a system from the perspective of specific system concern" [17]. The conventions of how an architectural view is constructed and interpreted are given through a corresponding architectural viewpoint. Several views on the architecture of the system-of-interest allow for factoring the design task into smaller and specialized tasks.

For a given concern, there exist several views at different levels of abstraction. A hierarchical design process allows for the co-evolution of requirements and architecture, known as the "twin peaks of requirements and architecture" [19, 20]. Based on ideas from compositional thinking, an evolution of system architectures seems possible by establishing suitable descriptions of the abstraction levels for the architectural views, their classification into clusters of concern, and the relation between the views. We call the framework "compositional" because it is built up from different "modules," called clusters of concern, at different levels of abstraction [18].

10.2.3 Clusters of concern

Clusters of concerns are determined through the identified use cases based on the operational context and high-level goals for the desired AI system. For example, privacy might not be of concern for an AI-based diagnostic system detecting faults of a welding robot, but safety could be of paramount concern. Four major groups of concerns emerged for the architecture framework:

Behavior and context contains aspects that concern the static and dynamic behavior of the system, as well as the context and constraints for the desired behavior. To describe an architecture reflecting the desired behavior of the system, two clusters of concern are introduced: **Logical Behavior** covers views that are concerned with the static behavior of the system, and **Process Behavior** covers views concerned with the dynamic behavior of the system. The **Context and Constraints** cluster of concern covers views on the system that define the context and limits the design domain for AI systems. For AI systems, it is beneficial, sometimes even required, to explicitly state the desired context and to define views on the constraints and the design domain of the system. An example is the Operational Design Domain of automated vehicles.

Means and resources contains aspects of the system that enable the desired behavior. The concerns in this group include views that allow to the description of the resources and means available for the system to execute the desired behavior in a given context.

Typical views include the hardware architecture and component design of the system under the cluster of concerned **hardware**. Additionally, three AI-related clusters of concern have been identified that are fundamental "means to execute a desired behavior."

First, the concerned **AI models** contain views that describe the setup and configuration of the required AI model, including the choice of the right deep-learning model. For example, the classification of objects in an optical video stream requires a different deep neural network configuration and then recognizing commands in a voice recording or predicting trajectories of other vehicles in the vicinity. Choosing the right AI model setup is a system design decision which requires suitable views on the AI model in relation to the overall system.

Furthermore, the learning strategy of the AI model has a paramount impact on the final behavior of the AI system. The **learning** cluster of concern covers views on the system that allows for defining and setting up the learning environment of the AI model. This can include the definition of training objectives and views that outline the chosen optimizer for training. Planning and preparing the learning of the AI model therefore becomes a "mean to execute a desired behavior" within an AI system. Learning can be conducted through preparing training datasets, or, in the case of reinforcement learning, could be done in a simulated environment.

Data strategy contains views that support collection and selection for training, validation, and runtime data of the AI model. Views can describe methods for data creation, data selection, data preparations, and runtime monitors of data used by the AI. Trained with the flawed datasets (e.g., bias present in the data), the behavior of the AI system will exhibit the flaws learned during the learning process (e.g., it will show a bias in the decisions). The concerns of an **AI model**, **Learning**, and **Data Strategy** have many dependencies on each other, which will be expressed through correspondence.

Communication deals with aspects of data, connectivity, and communication between nodes or components of the desired system, which is one major concern when developing distributed systems, such as automotive systems, or systems in the IoT. Communication is what drives the IoT. Two clusters of concerns have been identified: First, **Information** accumulates views on the system that model the information and data exchanged in and through the system-of-interest. Second, the cluster of concern **Connectivity** contains views on the means of communication available to the system and its resources.

Quality concerns basically encompasses all quality aspects described through quality attributes, which can affect the architecture of the system. Examples are safety, security, privacy, robustness, and ethical concerns. The latter can include aspects such as fairness and explainability. Recent legislation shows that ethical aspects become a central concern when developing AI systems [21]. This group contains concerns that influence the desired quality of the system. The cluster of concern **safety** provides an example here: Assume one is to follow the workflow of ISO 26262 [21]. The starting point to designing a safe system is to identify safety goals that the architecture, as part of the functionality-providing item, needs to fulfill. This is often done through a Hazard Identification and Risk Assessment (HARA), which provides abstract information applicable to the entire system. On the next lower level of abstraction, the functional safety concept provides a view of a more detailed system architecture that introduces functional safety requirements and redundancies (through safety decomposition in hardware and software components) with the aim to assure the fulfillment of the earlier specified safety goals. On the next more detailed level, the technical safety concept provides information on the technical realization of the functional safety concept. In addition, and not explicitly mentioned in ISO 26262, we propose that the runtime behavior and monitoring is part of the system design process.

For safety concerns, this could mean the introduction of safety degradation concepts and safety monitoring. Further identified relevant clusters of concerns for quality aspects of an AI system in the IoT are **Security**, **Privacy**, and **ethical aspects** such as Fairness and Transparency. For embedded systems, **Energy Efficiency** can be taken up as an explicit quality aspect covered by a separate cluster of concerns. Unlike previous architectural frameworks for the IoT, the compositional thinking in the architectural framework allows for co-designing the system to fulfill the explicitly identified quality concerns. It means that already early in the system development, correspondences between the views regarding the quality concerns and other views in the architecture description are established. The final system can then be said to be "Safe by design," "Secure by design," "Efficient by design," or "Fair by design."

Table 10.1 provides a list of viewpoints, which govern architectural views in the architecture framework, that we assume to be novel and relevant specifically toward the AI components of the system.

10.2.4 Levels of abstraction

The architectural views are not only sorted by clusters of concerns as discussed previously but also by their represented level of abstraction. We found it most beneficial to follow four levels of abstraction, specifically **knowledge and analytical level**, **conceptual level**, **design level**, and **runtime level**:

Knowledge and analytical level: The first level of abstraction includes architectural views that provide an abstract and high-level view of the system-of-interest. On that level, all views provide a way to describe the system and context on a knowledge level, which provides information for further, more concrete system development. For example, the high-level AI model view could elaborate on which functions should be fulfilled through an AI.

Conceptual level: On the next level of abstraction, the views provide a more concrete description of the overall system-of-interest. Components are not detailed yet, but the overall system composition becomes clear and the context of operation is clearly defined. For example, the AI model could be concretely shaped as a deep-learning network with a required amount of layers. All views on this level combined provide a system specification that sets the system-of-interest in context and elaborates on how the desired functionality is fulfilled.

Table 10.1 Description of clusters of concern in the framework.

Concern	Description
Behavior and Context	***Aspects that concern the static and dynamic behavior of the system, as well as the context and constraints for the desired behavior.***
Logical Behavior	Views that are concerned with the static behavior of the system.
Process Behavior	Views concerned with the dynamic behavior of the system.
Context and Constraints	Contains views on the system that define the context and limit the design domain.
Means and Resource	***Contains views on aspects of the system that enable the desired behavior.***
Hardware	Includes views on the hardware architecture and component design of the system.
AI models	Contains views that describe the setup and configuration of the required AI model. Views can include model design, for example, neural network setup or views detailing the configuration of the AI model.
Data strategy	Views that support collection and selection for training, validation, and runtime data of the AI model. Views can describe methods for data creation, data selection, data preparations, and runtime monitors of data used by the AI.
Learning	Covers views on the system that allows for defining and setting up the learning environment of the AI model. This can include the definition of training objectives and views that outline the chosen optimizer for training.
Communication	***Contains views of data, connectivity, and communication between nodes or components of the desired system.***
Information	Accumulates views on the system that model the information and data exchanged in and through the system-of-interest.
Connectivity	Contains views on the means of communication available to the system and its resources.
Quality Concerns	***Encompass quality aspects which can be described through non-functional requirements which affect the architecture of the system.***
Ethics	Views that regulate ethical aspects, such as fairness or transparency of the system.
Security	Views that ensure the security aspects of the system.
Safety	Contains views governing the safety aspects of the system. The views can stem from standards such as ISO 26262.
Energy Efficiency	This cluster of concerns contains views ensuring energy efficiency, especially for mobile devices.
Privacy	Here, views can be contained that ensure privacy requirements, such as for example requested by regulatory authorities.

Design level: The most concrete level at the design time of the system is the design level, which includes views that concretely shape the final system-of-interest. Resources are allocated to components, the AI model is configured to work most efficiently in the given environment, and the concrete component hardware architecture is defined. The solution specification describes the final embodiment of the system-of-interest.

Runtime level: Complex systems, both AI-driven and conventional, often require forms of monitoring and operations control. The purpose of runtime monitoring can be manifold: On one hand, monitoring a deployed system at a run time provides valuable feedback about its performance and reliability to developers and product owners. DevOps is an essential component of an agile development framework, and early detection of issues in a deployed system allows for a swift response from the developers. Furthermore, some requirements of the system might not be exhaustively testable before the deployment of the final system. This is especially the case for AI systems because we have to anticipate undesired behaviors of deployed AI algorithms. By constantly monitoring the decisions of the AI algorithm, such deviations from the intended behavior can be detected and mitigated, for example, through retraining or by "pulling the plug." Most AI systems are not "adaptive." They are trained and tested with a dataset representing the desired context in which the AI system is intended to operate in under the assumption of stationarity in the probability distribution of the data. In reality, the assumption of stationarity of the probability distributions does not hold in most cases, for example when the context, in which the AI operates, can change over time. Concepts like continual learning allow the AI to handle drifts in data distributions. However, continual learning requires runtime monitoring concepts to detect deviations from the currently learned context, and automatic data collection (and labeling) for autonomous retraining of the AI model. These aspects of changes in runtime behavior are described on the runtime level of abstraction in the compositional architectural framework.

The final conceptual model of a compositional architecture framework based on the stated propositions is illustrated in Figure 10.1.

10.2.5 Compositional architecture framework

Figure 10.2 presents a compositional architectural framework that includes all earlier identified concerns for distributed AI systems and all levels of abstractions for AIoT systems [18].

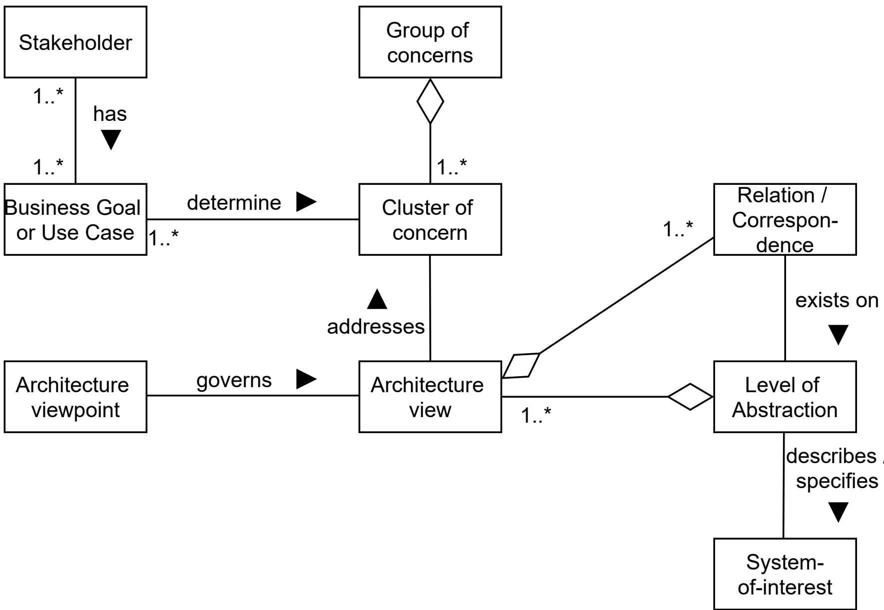

Figure 10.1 Conceptual model of a compositional architecture framework [18].

Business Goals and Use Cases													
	Behaviour and Context			***Means and Resources***				***Communication***		***Quality Concerns***			
	Logical Behaviour	**Process Behaviour**	**Context & Constraints**	**Data Strategy**	**Learning**	**AI Model**	**Hardware**	**Information**	**Connectivity**	**Ethics**	**Privacy**	**Security**	**Safety**
Analytical Level	Function components	Interaction	Context assumptions	Data ingestion	Training objectives	High level AI model	High level hardware architecture	Compilation	Interfaces	Ethic principles	Privacy impact analysis	Threat analysis (TARA)	Hazard analysis (HARA)
Conceptual Level	Logical components	Logical sequences	Context definition	Data selection	Training concept	AI model concept	System hardware architecture	Information model	Node connectivity	Ethic concept	Privacy concept	Cyber-security concept	Functional safety concept
Design Level	Computing ressource allocation	Resource sequences	Constraints / Design Domain	Data preparation / manipulation	Optimiser settings	AI model configuration	Component hardware architecture	Communication model	Resource connectivity	Ethic technical realisation	Technical solutions for privacy	Technical cyber-security concept	Technical safety concept
Run Time Level	Behaviour monitoring	Adaptive behaviour	Context monitoring	Runtime data monitoring and collection	Manage continous improvements	AI model performance monitoring	Hardware performance monitoring	Data monitoring	Connectivity monitoring	Assessment / auditing of AI decisions	Assessment of privacy compliance	Security monitoring / threat response	Safety monitoring / safety degradition

Figure 10.2 Compositional architecture framework for AIoT systems, categorizing views in different clusters of concerns on different levels of abstraction [18].

10.2.6 Applying a compositional architecture framework in practice

Based on the experience of applying a compositional architectural framework, the following guideline can be provided:

Step 1: Identify clusters of concern. Clusters of concerns are identified. Initially, larger groups of concerns (such as functionality, hardware, communication, and quality) can be defined, which are then refined into atomic clusters of concerns.

Step 2: Identify levels of abstraction. Levels of abstractions are identified. The number of required levels depends on the size and complexity of system-of-interest and the development settings of the company. Three to four different levels of abstraction seem a good default.

Step 3: Add existing architectural decisions. Known architectural decisions are entered into the matrix. Most development projects do not start from scratch but instead must reuse or integrate into existing architectures. Prior knowledge, such as an existing component architecture, can be entered into the appropriate clusters of concerns and level of abstraction in the architecture matrix.

Step 4: Add missing architectural views. Architectural views are added. Relations (morphisms) are created between the architectural views at each level of abstraction such that no inconsistencies occur when looking at the system-of-interest from different architectural views.

Step 5: Add missing relations. All relations between architectural views must be mapped onto corresponding views of the next lower level of abstraction. If a relation between two architectural views on a higher level of abstraction does not have a correspondence on the next lower level of abstraction, the relation might be unnecessary and can be removed, or a corresponding relation needs to be created.

Step 6: Iterate if needed. During the system development, additional clusters of concern might be discovered that iteratively are added.

Steps 1–5 are illustrated in Figure 10.3. At each step, implied requirements on aspects related to the corresponding architecture view are identified and derived.

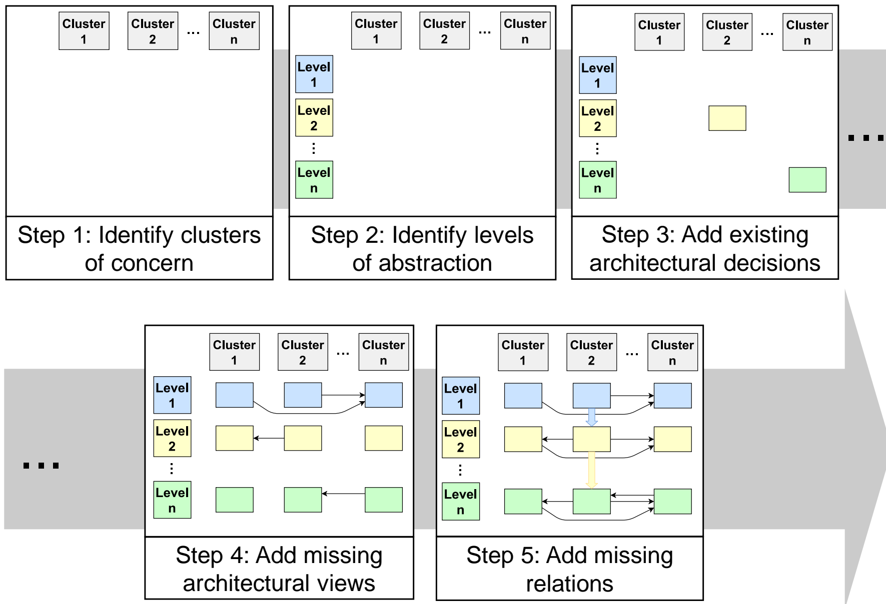

Figure 10.3 Steps taken for defining a compositional architectural.

10.3 WebAssembly as a Common Layer for the Cloud-edge Continuum

The cloud is an immense ecosystem of countless providers offering different virtualized services to supply an enormous demand for computer applications. Some of these applications ended up in the cloud to be more convenient or cheaper to maintain, others were initially built for the cloud for scalability and availability while relying on its naturally distributed and replicated nature. Clouds can also offer lower latency, more resiliency, or regulatory compliance. Regardless of the reason, **cloud computing** has probably become the most prominent infrastructure supporting applications today. With a growing number of multi-cloud software, dealing with heterogeneous cloud providers and technologies has become a common issue.

Telecommunication companies began deploying their own distributed infrastructure, installing small, cloud-like clusters closer to consumers of their services to improve performance, latency, or reliability. Local governments and other infrastructure providers such as energy and transportation followed suit, deploying their own small clusters of fairly powerful

computing devices close to the human activities they support. The use of these highly distributed devices is collectively known as **edge computing**.

Today's scenario is completed by billions of sensing and actuating devices deployed around the globe, referred to as the **Internet of Things** or IoT. Such devices often have limited processing capabilities and perform simple tasks like measuring a temperature or turning a lightbulb on and off. They are connected to the Internet, more than often coordinating their function through edge devices, and connecting users through cloud services.

The combined existing infrastructure of IoT, edge, and cloud form an abstraction that is currently being called the cloud-edge-IoT continuum, or simply the **cloud-edge continuum**. This collective infrastructure is anything but continuous, as each part exists in a separate silo, built of proprietary solutions, as shown in Figure 10.4. Developers of applications spanning over the continuum must implement specific solutions for each silo, often built with incompatible software components. The lack of a seamless environment makes it much more difficult to profit from the collective advantages of the continuum.

Finally, applications shared by multiple users always counted on some sort of **security**, usually dealing with encryption, authentication, and access control, and there are many established tools. With the advent of the cloud, which is accessed over the Internet, security has become a fundamental part of all applications. Edge-cloud continuum application vendors, developers, and users need to rely on the entire continuum – cloud, edge, and IoT – to ensure their data is secure and their calculations are accurate.

An ideal seamless cloud-edge continuum should provide a lightweight execution environment with a similar (or even identical) software and hardware interface that allows unmodified code to run on any machine in the

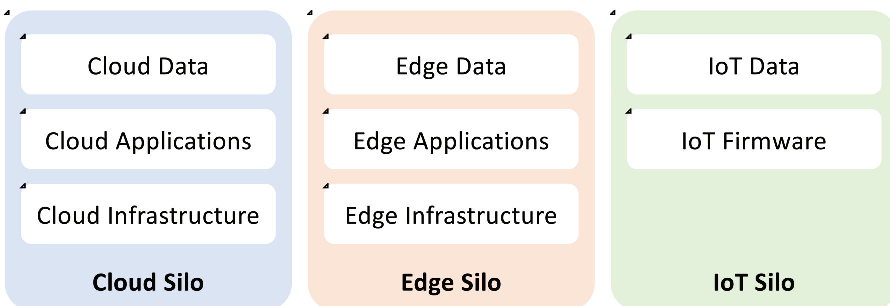

Figure 10.4 Independent cloud, edge, and IoT silos.

system. A typical cloud-only environment is already fairly complex, composed of several different hardware components, leveraged using extensive software components, managed by large engineering teams, and shared among many tenants. Adding edge and IoT to the picture shifts scale and heterogeneity to another dimension.

In this chapter, we propose using WebAssembly as the core component of a seamless environment spanning over the entire continuum. We advocate that the technology provided by WebAssembly is suitable for the implementation of applications on most hardware devices and software environments of the cloud-edge continuum, with the appropriate level of security. Modern hardware can execute WebAssembly with near-native code performance. Combined with special hardware features that guarantee the confidentiality and integrity of applications, WebAssembly abstracts the complexity of software development while providing a trusted environment. Naturally, as with any nascent technology, many parts needed to implement a seamless continuum are still missing.

In the sections that follow, we debate the drawbacks of existing software architectures in more detail. We then present WebAssembly and its benefits for implementing the continuum, in particular when supporting AIoT applications. We conclude with a few ideas for future work on the topic.

10.3.1 Building blocks of a seamless continuum for AIoT

There are already some initiatives for a common environment for cloud, edge, and IoT silos. In this section, we present a few popular ones and compare them to a solution using WebAssembly as proposed.

The java virtual machine (JVM) is one of the first practical implementations of common environments that address the problem of applications running on heterogeneous underlying systems. By and large, the JVM is one of the most comprehensive choices today, with implementations ranging from commodity servers to embedded devices. Still, the JVM supports very few programming languages and adds significant performance penalties compared to running C programs natively. Java programs depend on large numbers of class libraries, which imposes a large memory footprint for the execution of even the simplest programs.

Containers have recently emerged as an alternative to running applications in heterogeneous environments. They are, however, defined for specific architectures and a specific operating system interface, and recompilation is necessary to get containers that can run, for example, on Intel and Arm

devices (popular as cloud and edge devices, respectively). WebAssembly has the generality of JVM and the ease of use of containers, making it possible to build cross-platform software that runs with negligible performance losses and a small memory footprint.

Deploying applications automatically in a distributed system involves addressing aspects such as access control and resource management, as well as monitoring and optimizing computing and communication. We are not aware of any practical, specific tool that covers the entire cloud-edge continuum. We do not deal with this problem here, but we suspect it would be possible to adapt many of the existing tools designed for the cloud, assuming the underlying systems become more homogeneous. Also, some authors have already started working on models for integrating cloud and edge devices into one seamless deployment system [23, 24].

Security has already proven essential in standard cloud systems, where application users must have guarantees that the confidentiality and integrity of their data will be respected. These guarantees are difficult to provide in a multi-tenant system, where co-tenants can abuse the system's vulnerabilities to discover (or infer) someone else's application data. Also, one common deterrent for cloud adoption is the provider's curiosity, because they have all the administrative power needed to inspect all content across all physical machines. From an opposite point of view, providers want to be protected from malicious tenants who may want to exploit infrastructure vulnerabilities for their own benefit.

Compared to the cloud, edge infrastructure is much more distributed. Edge devices are installed in end-user buildings and other shared infrastructures, even in public spaces, making it impossible to maintain physical control over all the resources. Same as with the cloud, edge administrators have physical access and control of the edge devices they manage. But contrary to the cloud, edge users are close to the devices and can even abuse them physically. We believe that edge infrastructures offer far fewer security guarantees than the cloud.

Most current popular computer architectures include some form of trusted execution environments (TEEs). They allow code execution in an isolated part of the CPU, where access by other software is architecturally impossible. A TEE can run a program and protect its data so that a machine administrator cannot access it. Current implementations usually have an additional execution mode in the processor and may even offer memory encryption for TEE data. The currently most popular implementation of TEE is Intel's Secure Guard Extensions (Intel SGX), for which commercial cloud services such as

Azure Confidential Computing already exist. For edge and IoT deployments, the most popular architecture (Arm) offers TrustZone as a TEE. Again, proprietary and incompatible solutions in the underlying hardware make it difficult to reuse trusted software components from cloud to edge and vice versa.

Confidential containers could be a viable alternative for deploying applications on the continuum, as suggested by Scontain [25]. They are similar to traditional containers, except they run entirely in a trusted environment. However, like other containers, they are platform dependent. They are also expensive in terms of the resources required in many cases since they can contain significant amounts of operating system functions. Microsoft's Azure Sphere follows the same idea, offering a unified programming model and support for trusted execution technologies. But it only supports a few programming languages and relies heavily on other Microsoft services.

By proposing WebAssembly as an execution model combined with trusted execution environments, we can provide a seamless portability base for running trusted applications. The same base can be used to deploy applications on edge or cloud devices, with similar security guarantees. Also, previous work [26] has shown that a double-sided sandbox enabled by a WebAssembly TEE provides better security for the provider and for the tenants. In the context of AIoT, securing proprietary machine learning models is of utmost importance. Leveraging TEEs as a security mechanism to offload inference removes the burden of having pervasive communication to the cloud and lowers the number of end-user information to transfer offshore. As a result, AIoT systems are more autonomous, while better preserving the owners' privacy, which is an essential concern in the years to come.

Many different IoT infrastructures have been deployed and are already continuously generating data that feed cloud applications worldwide. Components in the application chains (IoT to edge to cloud) can be updated independently to add new functionalities and eliminate vulnerabilities. There is increasing usage of federated machine learning, where edge devices work together to build a model without revealing all the details of each user's data, helping to protect privacy. **Remote software attestation** [27], which is usually paired with TEEs, also plays a fundamental role in such a dynamic, distributed scenario. It makes it possible to build trust in certain software components and to check their authenticity and integrity. It also allows ensuring that one is remotely communicating with a specific, verified program. We believe that attestation plays an essential role in building a fully trusted environment for running cloud-edge continuum applications. Hence, cloud

applications can infer security guarantees from AIoT software using attestation, despite being in untrusted physical environments, and can delegate part of computations.

10.3.2 WebAssembly as a unifying solution

WebAssembly is a rather new and universal virtual instruction set architecture. Unlike previous cross-platform efforts such as Oracle's Java and Microsoft .NET, WebAssembly is being developed from the ground up by a consortium of open technology companies, including Microsoft, Google, and Mozilla. While originally designed to increase performance for active web pages, WebAssembly does not depend on web-related functionality and is increasingly used to build standalone applications. WebAssembly has many advantages to being used as a unified execution unit for the cloud-edge continuum. First, WebAssembly can be generated by compiling a variety of programming languages. Second, unlike Java and .NET, WebAssembly is lightweight, has minimal dependencies, and offers additional security benefits like sandboxing.

WebAssembly interacts with the operating system thanks to the WebAssembly System Interface (WASI), a standardized specification of a POSIX-like interface. It is designed with conciseness and portability in mind, allowing platforms to easily implement it, being ideal for constrained environments such as IoT and Edge devices and TEEs. Common compilers for languages like C and Rust seamlessly translate POSIX calls into WASI calls. In addition, WASI follows the concept of capability-based security, where access to each system resource must be granted by the runtime, such as file system or socket interactions, materializing a strong boundary between the applications and the operating system.

There are currently a few execution models for WebAssembly code: interpretation, just-in-time (JIT), and ahead-of-time (AOT) compilation. Runtimes like WAMR [28] can be adapted to offer one or more execution models, with different memory footprints (209 KiB for AOT, 230 KiB for interpretation, and 41 MiB for JIT). A growing list of toolchains (LLVM, Emscripten) already supports WebAssembly as a compile target for various source languages, including C, C++, and Rust, with other languages such as C#, Go, Kotlin, and Swift being under active development. For all these reasons, we believe WebAssembly is an attractive practical binary architecture choice to be used in the entire continuum.

10.3.3 The case for a TEE-backed WebAssembly continuum

Trusted execution environments aim to provide safe and trustworthy code execution on (remote) untrusted hardware. Hardware manufacturers have provided TEE implementations more than a decade ago, each one of them offering different features and security guarantees. The most influential TEEs that are currently marketed are Intel SGX [29], Arm TrustZone [30], and AMD Secure Encrypted Virtualization (AMD SEV) [31]. These technologies enable processing data in isolated memory areas that can neither be accessed nor tampered with by more privileged software, such as the operating system or the hypervisor. Hence, cloud providers and edge device owners with management rights or even physical control cannot access the data and computation of a tenant, protecting the confidentiality and integrity of their applications.

Cloud providers, such as Microsoft Azure and Google Cloud, already market confidential computing, and we expect widespread adoption of these services due to the demand driven by the cloud-edge continuum. We observe that the rich ecosystem of trusted environments largely varies in terms of security, threat models, and implementation. However, defining a common basis for trusted execution and making it widely available in both cloud and edge environments is essential for the continuum and the industry in general. For that reason, Arm, Intel, Microsoft, and others created the confidential computing consortium (CCC), supporting open-source projects for trusted execution technology under the umbrella of the Linux Foundation. A unified abstraction for TEEs in the cloud-edge continuum must take support and shape from such ongoing efforts. For that reason, the CCC is involved in many projects, such as Enarx [32] and Veracruz [33], which aim to provide WebAssembly support in TEEs independently from hardware.

In our previous work, we proposed a few solutions to execute general-purpose WebAssembly applications within TEEs. We developed Twine [34] to bring a WebAssembly runtime into Intel SGX enclaves, leveraging WASI to interact with the TEE facilities and the untrusted operating system. More recently, we proposed WaTZ [35], a trusted runtime for Arm TrustZone with added remote attestation. The latter, an essential feature for providing trust for remote applications, is surprisingly missing in Arm's architecture. We believe that industrial versions of our prototypes will help pave the way to build distributed applications on the cloud-edge continuum that providers, developers, and users can safely trust.

10.3.4 WebAssembly performance

We refer to our previous work for many experiments regarding WebAssembly performance. We first proposed a solution to run general-purpose WebAssembly applications inside Intel SGX TEEs, leveraging WASI to interact with the untrusted OS, while shielding the file system primitives to prevent eavesdropping. Later, we proposed a trusted runtime environment for Arm TrustZone with remote attestation of WebAssembly code. A more recent publication contains an extended version of this chapter, with some detailed performance figures [36]. We refer the reader to these publications for the full detail of our measurements.

In the performance measurements we made, we used WebAssembly inside TEEs to implement many frequent tasks done by useful programs. To measure the low-level cost of using WebAssembly, we used Polybench/C [37], a tool that implements several sorts of different programming language constructs frequently used, allowing us to compare the quality of different compilers. We observed similar performance losses when using WebAssembly on x86 and Arm architectures, with the execution time being increased by 30% on average.

To produce a comparison using more resources such as memory and disk, we compared the execution performance of SQLite, a widespread and embeddable database management system, as most real-world applications generate, store, and retrieve information to operate. As such, we used the built-in benchmarks of SQLite named Speedtest1 [38]. Each Speedtest1 experiment targets a single aspect of a database, such as selection using joins or the update of indexed records. In our evaluation, WebAssembly was almost three times slower than native code on an Intel x86 processor, and roughly two times slower in an Arm processor. Interestingly, since we made these performance comparisons at different moments in time, we could observe clear progress in the environment. WebAssembly was four times slower in the experiments we did two years earlier, using the same hardware and software, but with newer versions of the compiler and the runtime environment. These enhancements over the years strengthen the perspective of using WebAssembly as a universal, lightweight, yet versatile bytecode to enable platform independence across the continuum.

10.3.5 WebAssembly limitations

Although current compilers such as LLVM are mature enough to generate proper WebAssembly bytecode, the system call support currently offered by

WASI is rather limited. Extending WASI to be more POSIX-compliant would probably reduce the ability to use it in several, more protected, environments, such as web browsers. A different alternative is proposed by Emscripten, which directly translates the source code into POSIX functions and system calls. This helps to run older WebAssembly programs on POSIX systems with only a few modifications, but it reduces the portability. We note that the WebAssembly subgroup that focuses on standardizing WASI thoughtfully extends the specifications to be features-complete.

Running WebAssembly code incurs a performance overhead. Some programs can run up to three times slower than their native version, depending on the type of workload. This can be explained by many factors, such as increased register pressure, additional branch instructions, increased code size, stack overflow checks, and indirect call checks. While some of these issues can be compensated for by having compilers spend more time generating better code, other factors are a consequence of WebAssembly's design limitations, which would require changes in its specifications, at the cost of making it more difficult to implement.

WebAssembly uses linear memory to store the heap of a running program, with a limited number of 64KiB pages, for a total of 4GiB. While most software will not require more than this amount of linear memory, this may limit some server-side applications, such as training large deep-learning models or keeping large databases in memory. Recent proposals aim to extend this limit by increasing the number of allocable pages, raising the theoretical memory ceiling to 16 EiB (64-bits wide).

As with any young technology, WebAssembly still needs more efficient implementations for many useful features. Future contributors may suggest WebAssembly and WASI extensions to relax the constraints or extend the capabilities of the specification. For example, WASI-nn proposes adding a WASI machine learning module to facilitate model inference. We also anticipate that many current limitations for the cloud-edge continuum will disappear thanks to compiler advances, specification extensions, and better WebAssembly support for popular requirements.

10.3.6 Closing remarks concerning the common layer

It is impossible to precisely predict which will be the winning technology used to build the cloud-edge continuum. Yet, we envision it as an interoperable, scalable, and distributed system in which any piece of software can reside on any device, regardless of the underlying platform. Such capabilities will

change the development lifecycle of future applications, allowing developers to focus on business value rather than spending time with the complexity of each individual piece of infrastructure. WebAssembly is perfectly suited to this task thanks to its abstraction of the operating system, device type, programming language, and the additional security guarantees it can offer with TEEs.

We briefly presented some performance results showing that WebAssembly is a viable alternative to running native applications, with acceptable overhead. We have covered many aspects of successfully adopting WebAssembly to implement the cloud-edge continuum. Many challenges remain to be overcome, such as improving interoperability with existing programming languages and extending WASI to better support more complex applications. Also, much progress is still necessary for terms of middleware, which connects the components of the continuum and simplifies the deployment and migration of applications. Thanks to the experience we acquired with WebAssembly and Trusted Computing ecosystems, we are confident that they are a well-suited software development foundation for building large-scale systems such as the cloud-edge continuum.

10.4 TOCTOU-secure Remote Attestation and Certification for IoT

A key component in securing connected IoT systems is ensuring the integrity of the IoT software-state and detecting any change. This is typically achieved with remote attestation (RA), which aims at verifying the state of the software/memory of an untrusted attester (i.e., an IoT device) by allowing a trusted verifier to engage in a challenge-response-based exchange of proof. RA mechanisms rely on hardware/software/hybrid Root-of-Trust. As a result of said attestation, the attester is certified with a certain level of assurance guaranteeing software-state integrity that impacts trust decisions within networked systems. The attestation often results in software updates or issuing certificates indicating device assurance levels. The certificates include information like the assurance evidence, device IDs, assurance level indicating the trustworthiness of the device, etc. This assurance certificate only guarantees that an IoT device has a verified software stack. IoT devices also need conventional X.509 certificates when strong authentication is required, which is enabled by public key infrastructure (PKI). There are efforts to bring conventional PKI to IoT [39–41], which meet IoT limitations such as resource

constraints of the device, the dynamic operational environment, diversity in the supply chain, etc.

It is important that we do not define yet another certification infrastructure for assurance certification, and integrate assurance certificates with existing state-of-the-art PKI. This chapter addresses both of these problems: (i) providing digital certification for device assurance (ii) as well as integrating the new assurance certificates into the existing PKI certification, without compromising the standard compliance and without security properties.

More specifically, in this chapter, we introduce and detail AutoCert (**Auto**mated digital **Cert**ification) to provide TOCTOU security by combining Remote Attestation results about assurance of device health with standard public key infrastructure (PKI) authentication processes.

In the context of RA and certificates that reflect the attested state of the device, the time-of-check to time-of-use (TOCTOU) race condition may take effect. The time-of-check to time-of-use invalidity is a highly contextual problem, existing in remote attestation, operating systems, certifications, etc., and remains possible in this case as well. Due to the dynamic nature of IoT systems, the software state of the device may have changed in the delta time between the RA and the certificate issuance due to a software update, vulnerability exploitation, or software version update. Although potential solutions exist to prevent and resist TOCTOU attacks in trusted platform module (TPM)-based remote attestation, a solution that provides a mechanism to validate the current software-state against the attested state and use an assurance certificate without invoking RA again, is missing, and is critical in the IoT domain. However, a solution that provides a mechanism to validate the current software-state against the attested state in certificates without invoking RA again is also critical in the IoT domain.

10.4.1 AutoCert – proposed mechanism

The AutoCert mechanism is an automated procedure comprising interactions among an **IoT owner**, **IoT devices** as a part of a networked system, a trusted third-party responsible for attesting the device's software-state, for example, a Conformity Assessment Body (**CAB**), and a standard Certification Authority (**CA**) to enroll device certificates.

10.4.1.1 Pre-deployment

The manufacturer commissioned the IoT device with software, platform/device certificate, a dedicated TPM 2.0 chip, and a secure unique device

identifier during device initialization. The platform certificate binds the TPM to the IoT device. The secure unique device identifier, that is, UDevID, is a hardcoded identity like a device URI, EUI, or DevID playing a role in IoT device identification in local and global networks.

The TPM's Root-of-Trust originates with a unique 2048-bit RSA key pair, known as the endorsement key (EK). The TPM restricts the use of the EK to a limited set of decryption operations as per the TCG rules, and it cannot be used directly for device authentication or digital signatures. Therefore, we generate a 2048-bit RSA key pair, the attestation key (AK), using the EK as a seed for attestation. The attestation certificate (CertAK) corresponding to the AK is also generated at this state by the IoT manufacturer. The IoT manufacturers and solution providers classify IoT devices into usage profiles based on their deployment scenario, for example, smart home, automotive, industrial, critical infrastructure, smart grid, etc. In AutoCert, the IoT owner assigns a device_profile to the IoT device to enable security policies for devices within a network. This categorization assists CAs and CABs in conducting reasonable risk assessment and vulnerability management throughout the device lifecycle. The IoT device is configured to boot with trusted software that measures (i.e., calculates the hash) the next software to be run and stores this hash in a platform configuration register using the TPM2_PCR_Extend function. This process continues through the OS kernel code resulting in a chain of measurement. In AutoCert, we propose configuring security-critical software, libraries, files, and executables as a part of this chain of measurements.

10.4.1.2 Remote attestation

AutoCert's remote attestation is built on the challenge/response interaction model from the RATS architecture. Before a device is attested (Figure 10.5), the IoT owner is responsible for generating the reference values corresponding to the device software/s and securely transferring them to the verifier. We assume a confidential exchange of these values. Before the remote attestation begins, the IoT owner sends a signed request to the CAB with the **UDevID** and device_profile of the IoT device. The CAB sends a signed attestation request containing a random nonce N and a **PCRSelection** is sent to the IoT device. The TPM2_Quote function is used to generate the evidence. The cryptographically strong random nonce N uniquely distinguishes the evidence, determines its freshness, and prevents replay attacks. We propose the generation of an integrity key pair, IK, by the IoT device and sending it along with the evidence for the creation of an integrity_proof. The IK is an

RSA key-pair, IK_{priv}, and IK_{pub} generated with the TPM2_Create function using the **PCRSelection**. Since any change in the security-critical software on the device is recorded with an update to the PCR using TPM2_PCR_Extend, the use of this **PCRSelection** in creating the IK ensures that this key will not be valid if the software-state of the device changes.

A valid TPM-generated attestation key, the *AK*, is used to sign TPM-generated evidence. It serves as a way for third parties to validate keys and data generated by a specific TPM on an IoT device. On receiving the evidence, the CAB validates the accompanying signature and compares the evidence against reference values. Following the attestation result and using a suitable risk assessment mechanism (not discussed in this work), the attester's assurance level is calculated against the device_profile. The results of attestation and the assurance level are used by the CAB to ensure the software-state integrity

10.4.1.3 TOCTOU and integrity_proof

The integrity key pair, IK, is proposed to address the TOCTOU race condition. The **PCRSelection** contains the measurements computed and stored during the measured boot, representing the IoT device's software-state.

Using these PCRs in RA and generating the IK_{priv} and IK_{pub} key pair creates a dependence of the IK on the software-state of the device. As soon as the software-state changes due to a new vulnerability or malicious update, the IK is invalidated.

This forms the core of AutoCert procedures and is a part of the proof of the IoT device's software-state integrity, as it strictly locks the IK to a valid state of the device. We compute an integrity_proof by aggregating the value of **PCRSelection** used in evidence generation, that is, **PCRIntegrity** and the IK_{pub}. The integrity_proof , assurance level, and **UDevID** are then shared with a trusted CA. The CA now possesses records of attested IoT devices against their **UDevID** and assurance attributes. These attributes are integrated with the IoT profile of the standard X.509 certificate using custom extensions. This certificate $Cert_{AC}$ reflects a CA-verified device identity (authentication) as well as the CAB-attested software-state of the IoT device (assurance).

10.4.1.4 Verification for TOCTOU security

The verification of this integrity_proof for TOCTOU security applies to all IoT devices using X.509 certificates for authentication and establishing secure DTLS communication sessions with clients.

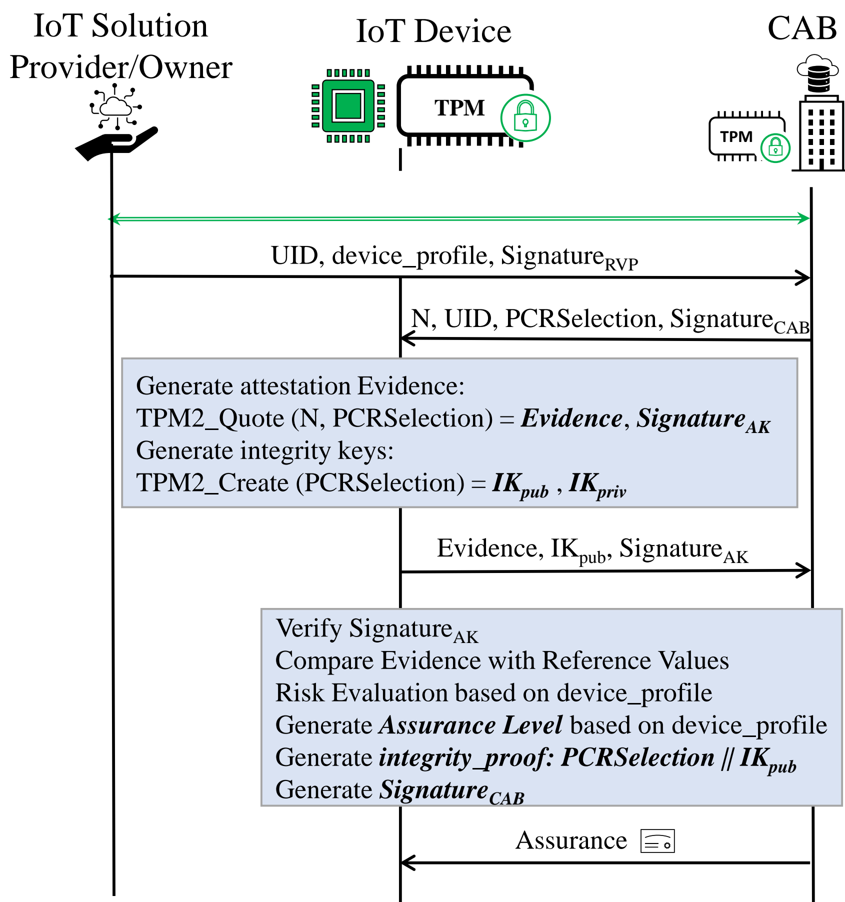

Figure 10.5 Remote attestation procedure.

To achieve assurance of the IoT device's software-state, the client performs two levels of integrity checks, as presented in Figure 10.6.

The first level of integrity check includes verifying the assurance level stated in the $Cert_{AC}$. This assurance level would form the basis of network access policies or authorization to access system resources.

However, as stated earlier, it is possible that the IoT device's software-state changes after the remote attestation process, or $Cert_{AC}$ enrollment. This can happen due to malware or vulnerabilities in existing software. This scenario presents itself as an instance of a TOCTOU attack, and checking the assurance level is insufficient in security-critical cases.

To eliminate this TOCTOU condition, AutoCert facilitates another level of integrity verification. To perform this Level2 integrity check, AutoCert introduces a lightweight service to ensure that the integrity_proof is valid. The verification process includes sending a random challenge by the client to the IoT device after signing it using the IK_{pub} from integrity_proof in the $Cert_{AC}$. Since the integrity_proof is locked to the state of the IoT device attested by CAB, it can only be decrypted by the IoT device if it possesses IK_{priv}, hence guaranteeing proof of possession. The IoT device decrypts the challenge, includes the current value of the **PCRIntegrity**, and signs it. The challenge ensures the freshness of this message exchange. The current value of the PCR concatenated with the challenge is received by the client, which verifies it against the PCR values from the integrity_proof, that is, the **PCRIntegrity** confirming that no changes have occurred concerning the software-state since attestation.

10.4.2 Implementation and experimental evaluation

As a proof-of-concept (PoC), we implemented the AutoCert setup with an attestation service on the IoT device, which is invoked when it receives an attest request. We also implemented an integrity verification service corresponding to the two levels of integrity checks. The experiments are performed using the OPTIGA TPM Evaluation Kit. The evaluation hardware is comprised of a Quad Core 1.2GHz, 64-bit Raspberry Pi 3 with 1 GB RAM and an Iridium board with OPTIGA SLM 9670 TPM 2.0. We choose TPM SLM 9670 for this evaluation since it is specially designed for use in automotive/industrial applications. The following set of experiments aims to measure the system-wide execution time of the proposed mechanism during different phases. We measured the round trip time (RTT) as the time elapsed from the start of each AutoCert phase until the completion of the phase. We measured the phases using a system clock in nanoseconds and iterated the experiments five to ten times to ensure statistical accuracy.

Phase 1 of AutoCert begins with a request to the CAB to initiate AutoCert remote attestation with the IoT device. The RTT of this phase is 28,800 ms. This phase is expected to execute during device assembly after the unique device keys are integrated into the hardware, and device software is installed. This does not interrupt runtime services like mutual authentication, where excessive delays disrupt services, timeout, or cancellation of operations.

Phase 2 of AutoCert is the certificate enrollment. On receiving a certificate enrollment request from the IoT device, the CA checks for assurance

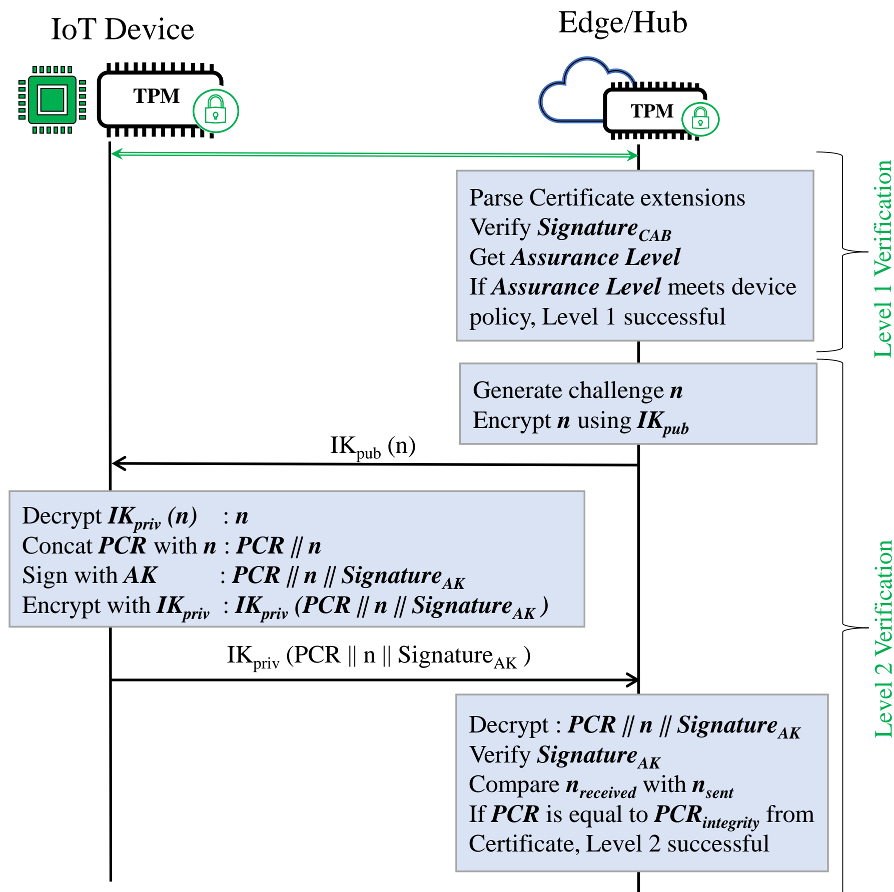

Figure 10.6 Verification procedure.

attributes received from the CAB, associated with the **UDevID** of the IoT device, and enrolls the certificate, including the assurance attributes. The enrollment of Cert_{AC} with assurance attributes takes 7104 ms. This measurement merely gives an estimate of the generation of a certificate with additional extensions. In actual events, certificate issuance and enrollment time also vary depending on the computational capabilities of the CA and network capacity.

Phase 3 of AutoCert provides 2 levels of assurance to the communicating devices. The proposed level 1 integrity check attains a basic level of assurance. This begins by verifying the signature and the assurance level from the Cert_{AC}. An extended TOCTOU security of assurance is provided

in level 2. The level 1 verification steps are executed in 0.7 ms, and the RTT for level 2 verification, including minor network delays between the two involved entities, is 4746 ms. The majority of the execution time during level 2 verification can be traced to the creation and loading of the encryption key. As these operations depend on the implementation of TPM specifications and adjacent function libraries, it is reasonable to state here that the RTT for level 2 verification is justified considering the hardware security guarantees provided by the TPM.

10.4.3 AutoCert – conclusion

This chapter presented AutoCert, addressing TOCTOU security in integrity certificates corresponding to software-state assurance in IoT devices and providing a standardized mechanism to distribute integrity certificates. AutoCert's remote attestation is based on IETF RATS relying on TPM2.0 for evidence generation. We have proposed the integration of the AutoCert mechanisms into existing standards to facilitate its adoption in the emerging PKI for IoT.

10.5 Conclusion

In this chapter, a compositional architectural framework was derived during focus groups within the project consortium. Compositional thinking allows for an effective co-design of all relevant concerns of the system-of-interest. Especially for AI components, the architectural framework allows for effective data selection, AI model development, and hardware design. Qualitative aspects, such as safety, security, and privacy, but also ethical aspects are explicitly considered throughout the design process. Furthermore, to ensure functionality and quality aspects of the system, the architectural framework considers monitoring concepts for runtime operations of the system.

In addition, a common layer for the cloud-edge continuum based on the WebAssembly virtual instruction set architecture is introduced. We discussed the historical context and the shortcomings of existing software development environments and shed light on what improvements can be implemented to arrive at seamless, secure applications across the continuum. We then presented WebAssembly's advantages for such applications, along with its preliminary performance comparison for executing benchmark payloads, thus supporting the concept's viability for building the unified technology.

Furthermore, we presented the time-of-check to time-of-use challenges that remote attestation and certification face in the context of AIoT systems. An overview of the seriousness and specificity of TOCTOU problems for IoT devices, resulting from resource constraints of such devices, was given, describing their operational environment, supply chain, vulnerability management, and others. Then, we highlighted the importance of developing a solution capable of software validation appropriate for IoT devices and described AutoCert as a proposed mechanism.

Acknowledgement

This publication incorporates results from the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 957197.

References

[1] Joel Höglund, Samuel Lindemer, Martin Furuhed, Shahid Raza. PKI4IoT: Towards Public Key Infrastructure for the Internet of Things. Computers & Security journal (Elsevier), Volume 89, Pages 101658, February 2020

[2] Joel Höglund, Martin Furuhed, Shahid Raza. Lightweight Certificate Revocation for Low-power IoT with End-to-end Security. Journal of Information Security and Applications (Elsevier), Volume 73, 103424, March 2023

[3] Joel Höglund, Martin Furuhed, Shahid Raza. Towards Automated PKI Trust Transfer for IoT. The 3rd International Conference on Public Key Infrastructure and Its Applications (PKIA 2022), September 9-10, 2022.

[4] Anitha Murugesan, Sanjai Rayadurgam, and Mats Heimdahl. Requirements reference models revisited: Accommodating hierarchy in system design. Proceedings of the IEEE International Conference on Requirements Engineering, 2019-September:177–186, 2019.

[5] Nalchigar, S., Yu, E., Keshavjee, K., 2021. Modeling machine learning requirements from three perspectives: a case report from the healthcare domain. Requirements Engineering 26, 237–254.

[6] Patrizio Pelliccione, Eric Knauss, Rogardt Heldal, S. Magnus Ågren, Piergiuseppe Mallozzi, Anders Alminger, and Daniel Borgentun. Automotive Architecture Framework: The experience of Volvo Cars. Journal of Systems Architecture, 77:83–100, 2017.

[7] Bosch, Jan, Helena Holmström Olsson, and Ivica Crnkovic. "Engineering ai systems: A research agenda." *Artificial Intelligence Paradigms for Smart Cyber-Physical Systems* (2021): 1-19.

[8] Lucas Bernardi, Themis Mavridis, and Pablo Estevez. 150 successful machine learning models: 6 lessons learned at Booking.com. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1743–1751, 2019.

[9] Phillippe Kruchten. Architecture blueprints—the "4+1" view model of software architecture, volume 12. ACM Press, New York, New York, USA, 1995.

[10] Eoin Woods. Software Architecture in a Changing World. IEEE Software, 33(6):94–97, 2016.

[11] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., Dennison, D., 2015. Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems 2015-January, 2503–2511.

[12] Henry Muccini and Karthik Vaidhyanathan. Software Architecture for ML-based Systems: What Exists and What Lies Ahead. Proceedings of the 43rd International Conference on Software Engineering,, mar 2021.

[13] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. How does Machine Learning Change Software Development Practices? IEEE Transactions on Software Engineering, 2020.

[14] Greg Giaimo, Rebekah Anderson, Laurie Wargelin, and Peter Stopher. Will it Work? Transportation Research Record: Journal of the Transportation Research Board, 2176(1):26–34, jan 2010.

[15] Anitha Murugesan, Sanjai Rayadurgam, and Mats Heimdahl. Requirements reference models revisited: Accommodating hierarchy in system design. Proceedings of the IEEE International Conference on Requirements Engineering, 2019-September:177–186, 2019.

[16] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. Software Engineering for AI-Based Systems: A Survey. Preprint, 1(1), 2021.

[17] International Organization for Standardization. ISO / IEC / IEEE 42010:2012: Systems and software engineering — Architecture description. Swedish Standards Institute, Stockholm, swedish standard edition, 2012.

[18] Heyn, Hans-Martin, Eric Knauss, and Patrizio Pelliccione. "A compositional approach to creating architecture frameworks with an application

to distributed AI systems." *Journal of Systems and Software* (2023): In print.

[19] Bashar Nuseibeh. Weaving Together Requirements and Architectures. Computer, 34(3):115–119, 2001.

[20] Jane Cleland-Huang, Robert S. Hanmer, Sam Supakkul, and Mehdi Mirakhorli. The twin peaks of requirements and architecture. IEEE Software, 30(2):24–29, 2013.

[21] European Commission. Regulation of the European Parliament and of the Council laying down harmonised rules on Artificial Intelligence (Artificial Intelligence Act) and amending certain Union Legislative Acts, 2020.

[22] International Organization for Standardization. ISO 26262:2018: Road vehicles— Functional safety. International Organization for Standardization, Geneva, 2018.

[23] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, et al., 'The Internet of things, fog and cloud continuum: integration and challenges', Internet of Things, vol. 3, pp. 134–155, 2018.

[24] Daniel Balouek-Thomert, Eduard Gibert Renart, Ali Reza Zamani, et al., 'Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows', International Journal of High Performance Computing Applications, vol. 33, num. 6, pp. 1159–1174, 2019.

[25] Sergei Arnautov, Bohdan Trach, Franz Gregor, et al., 'SCONE: secure Linux Containers with Intel SGX', 12th Symposium on Operating Systems Design and Implementation, USENIX, 2016.

[26] David Goltzsche, Manuel Nieke, Thomas Knauth, et al. 'AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting', 20th International Middleware Conference, ACM, 2019.

[27] Jämes Ménétrey, Christian Göttel, Anum Khurshid, et al., 'Attestation mechanisms for trusted execution environments demystified', 22nd IFIP International Conference on Distributed Applications and Interoperable Systems, Springer, 2022.

[28] WebAssembly micro runtime (WAMR), https://github.com/bytecodealliance/wasm-micro-runtime.

[29] Victor Costan and Srinivas Devadas, 'Intel SGX explained', IACR Cryptology ePrint Archive, 2016.

[30] Arm, 'Introducing Arm TrustZone', https://developer.arm.com/ip-products/security-ip/trustzone, 2019.

[31] Advanced Micro Devices. ‘Secure Encrypted Virtualization API: Technical Preview’, tech. rep. 55766, 2019.
[32] Enarx, https://enarx.io.
[33] Veracruz, https://veracruz-project.com.
[34] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, et al., ‘Twine: an embedded trusted runtime for WebAssembly’, 37th International Conference on Data Engineering, IEEE, 2021.
[35] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, et al., ‘WaTZ: a Trusted WebAssembly runtime environment with remote attestation for TrustZone’, 38th International Conference on Distributed Computing Systems, IEEE, 2022.
[36] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, et al., ‘WebAssembly as a common layer for the cloud-edge continuum’, 2nd Workshop on Flexible Resource and Application Management on the Edge, 2022.
[37] Louis-Noël Pouchet et al., ‘PolyBench/C the polyhedral benchmark suite’, 2018.
[38] Lv Junyan, Xu Shiguo, and Li Yijie, ‘Application research of embedded database SQLite’, International Forum on Information Technology and Applications, IEEE, 2009.
[39] Joel Höglund, Samuel Lindemer, Martin Furuhed, Shahid Raza. PKI4IoT: Towards Public Key Infrastructure for the Internet of Things. Computers & Security journal (Elsevier), Volume 89, Pages 101658, February 2020
[40] Joel Höglund, Martin Furuhed, Shahid Raza. Lightweight Certificate Revocation for Low-power IoT with End-to-end Security. Journal of Information Security and Applications (Elsevier), Volume 73, 103424, March 2023
[41] Joel Höglund, Martin Furuhed, Shahid Raza. Towards Automated PKI Trust Transfer for IoT. The 3rd International Conference on Public Key Infrastructure and Its Applications (PKIA 2022), September 9-10, 2022.