VEDLIoT
Very Efficient Deep Learning in IoT

ICT-56-2020 — Next Generation Internet of Things

# D5.4
# Integrated and extended Security, Safety and Robustness mechanisms and tools

Version 1.0

| Document Information | |
|---|---|
| Contract Number | 957197 |
| Project Website | `https://vedliot.eu/` |
| Dissemination Level | PU (Public) |
| Nature | R (Report) |
| Contractual Deadline | 31 January 2024 |
| Author | António Casimiro (FC.ID) |
| Contributors | Jämes Ménétrey (UNINE), Marcelo Pasin (UNINE), Sébastien Vaucher (UNINE), Carina Marcus (MAGNA), Alysson Bessani (FC.ID), José Cecílio (FC.ID), Tiago Carvalho (FC.ID), Bakary Badjie (FC.ID), Piotr Zierhoffer (ANT), Simon Bouget (RI.SE), Anum Khurshid (RI.SE), Peter Jonsson (RI.SE), Joakim Eriksson (RI.SE) |
| Reviewers | Mario Porrmann (UOS), Pedro Trancoso (Chalmers) |

The VEDLIoT project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 957197.

## Change Log

| Version | Date | Description of Change |
|---|---|---|
| 0.1 | 2023-09-11 | Document template created. |
| 0.2 | 2023-10-19 | Added contributions from MAGNA. |
| 0.3 | 2023-12-05 | Added contributions from FC.ID. |
| 0.4 | 2023-12-05 | Added contributions from UNINE. |
| 0.5 | 2024-01-05 | Added contributions from Antmicro. |
| 0.6 | 2024-01-15 | Added more contributions from UNINE. |
| 0.7 | 2024-01-24 | Integrated version with initial and final sections. |
| 1.0 | 2024-01-30 | Complete version, with added contributions from RI.SE, and addressing comments provided by reviewers. |

# Table of Contents

## List of Figures

# List of Tables

# Executive Summary

This deliverable, D5.4 “Integrated and extended Security, Safety and Robustness mechanisms and tools”, is the final deliverable of VEDLIoT’s Work Package 5 (WP5). During the project, WP5 was focused on security, safety and robustness aspects of AI-enabled critical automated systems. The deliverable provides final descriptions of the several components that were developed in WP5 during the project, including evaluation results where appropriate.

The deliverable is organized as follows:

Chapter 1 provides an introduction to the deliverable, explaining how the provided contents are aligned with the WP objective and the planned tasks.

Chapter 2 focuses on solutions for secure communication and execution on IoT platforms, by exploiting the possibility of using Trusted Execution Environments (TEEs), and dealing with WebAssembly applications. More specifically, the chapter describes Twine, which provides a WebAssembly runtime for Intel SGX, presents Fortress, a framework using ARM TrustZone to protect IoT devices, and also presents a solution for secure Publish/Subscribe communication, implementing an MQTT broker inside an Intel SGX enclave.

Chapter 3 describes an extension to previous work on Remote Attestation of IoT devices, now addressing Trusted Certification using hardware-based root-of-trust.

Chapter 4 describes work done in relation to the Contiki-NG OS, which is widely used in IoT devices. More specifically, it describes the efforts done to port the Contiki-NG OS to an FPGA platform that can run a RISC-V implementation, and the implementation of the newly standardized lightweight authenticated key exchange protocol called EDHOC (Ephemeral Diffie-Hellman Over COSE) standard for the Contiki-NG OS.

Chapter 5 presents a performance evaluation of the Trusted Verifier Service (SIRE), which is done in the context of three use cases involving edge devices.

Chapter 6 is then focused on simulation and testing of embedded and IoT systems, which is done using the Antmicro’s Renode open source simulation framework. The chapter wraps up the work done in the project concerning these aspects, describing latest updates and improvements of some Renode components.

Chapter 7 describes latest work aiming at increasing the robustness of machine learning, which is important for the reliability and safety of critical IoT systems. This includes the description of methods for the robustness verification of neural networks, for mitigating adversarial data poisoning attacks, and for handling inputs drifts in image classification systems using deep neural networks.

Chapter 8 turns attention to safety aspects, discussing the possibility of using shared data in traffic scenarios for improving safety and flow. The potential benefits come with challenges, from data generation to data distribution and processing, which are discussed in the chapter.

Finally, Chapter 9, sums up the work done in WP5 and provides concluding remarks.

# 1 Introduction

The main objective of **VEDLIoT** (Very Efficient Deep Learning in IoT) was to develop a platform for the Next Generation IoT architecture, which considers the full continuum of computation ranging from the far-edge to the cloud. The VEDLIoT platform can be used in various application domains, including autonomous driving, smart industry and smart home.

Given the substantial volume of data collected and the demanding computational requirements inherent in these applications, the need for efficient solutions is evident. In addressing this challenge, VEDLIoT leverages Artificial Intelligence (AI) and Deep Learning (DL) to effectively manage the inherent complexity of the problem. In addition, these systems and applications are often safety-critical, and deal with sensitive data then needs to be protected. Therefore, ensuring **security, safety, and robustness** is of utmost importance, and this is the focal point addressed in Work Package 5 (WP5). An overall perspective of VEDLIoT, including its major components and the security and safety aspects addressed in WP5 is provided in Figure 1.1 (right-hand side).

This deliverable is the last one in WP5 and hence provides a perspective of the final achievements concerning the definition, development and evaluation of mechanisms, methods and tools for increased security, safety and robustness. It is organized in several chapters, each of which groups together and describes work related to a common topic. For instance, Chapter 2 focuses on how to secure communication and computation when using IoT platforms, whereas Chapter 7 groups together the description of a set of methods developed in VEDLIoT to increase ML robustness. In the following five sections, each corresponding to a WP5 task, we describe how the work presented in the several chapters contributes towards the goals of these tasks.

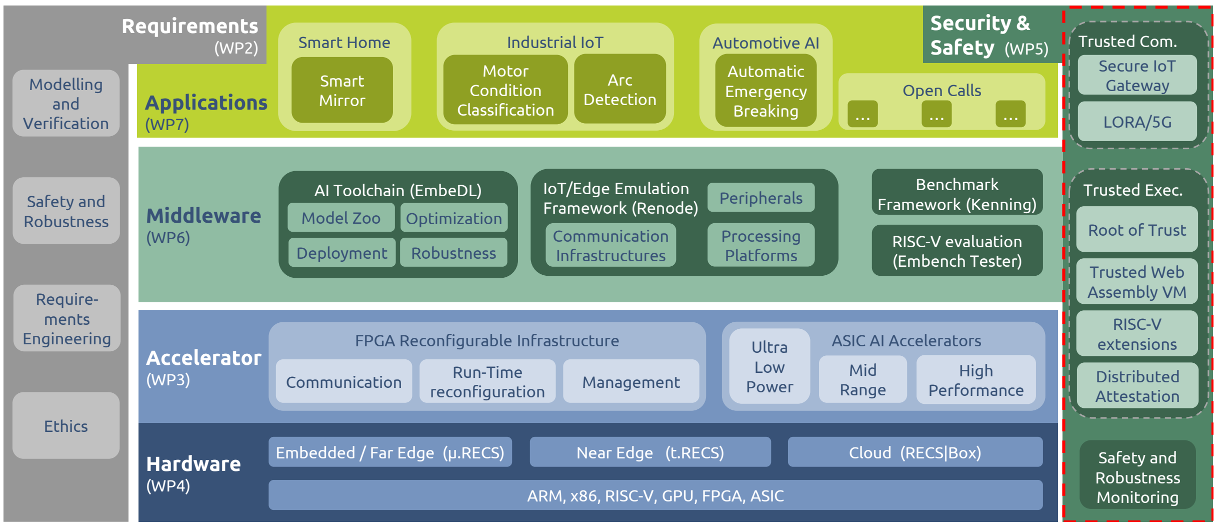


*Figure 1.1. Overall perspective of VEDLIoT.*

## Task 5.1: End-to-end attestation of distributed trusted environments

The main purpose and objective of this task was to develop end-to-end trust through a distributed attestation mechanism. To this end, we developed SIRE (truSted verIfieR sErvice), whose principles were introduced in Deliverable D5.1 [145] (there we referred to a Trusted Membership Service), and whose description, implementation and initial evaluation were presented in Deliverables D5.2 and D5.3. SIRE provides distributed Byzantine Fault-Tolerant (BFT) attestation and in this deliverable we present additional work on SIRE. In concrete, additional evaluation results are presented in Chapter 5. Furthermore, in this task we also provided a solution to address the well-known TOCTOU (Time-Of-Check to Time-Of-Use) problem, initially described in Deliverable D5.2 [100], and developed in Deliverable D5.3 [202]. Chapter 3 describes how to realise the proposed certification solution to assure trusted certification of IoT devices.

The evaluation of SIRE's performance herein presented has now been done in the context of three example application scenarios, which are directly or closely related to VEDLIoT use cases: a) membership management of IoT devices; b) autonomous vehicle coordination; and c) federated machine learning. All these examples involve a possibly large number of end devices that must be attested, so that they are recognized as trustworthy and can take part of some distributed computations. In fact, the first scenario is highlighted on a demonstration that has been prepared in connection to the VEDLIoT motor monitoring use case, involving SIRE and the secure MQTT broker described in Section 2.3. However, in the demonstration only one device is attested. In contrast, in this deliverable we provide performance results showing that operations supported by SIRE can scale to hundreds or even thousands of clients interacting with the service. Therefore, we show that distributed BFT attestation is perfectly feasible in realistic environments.

## Task 5.2: Security support for distributed execution and communication

In VEDLIoT, we committed to improve security at the edge by making extensive use of Trusted Execution Environments (TEEs). Either using Intel's SGX or ARM's TrustZone, we worked during the whole project duration towards creating solutions to exploit TEEs in order to run critical software in a secure way, and to make sure that it would be possible achieve end-to-end communication security. Chapter 2 is entirely devoted to present the latest achievements towards the objectives of Task 5.2.

One important option we took right from the start of the project was to use WebAssembly as an enabler, to ease the deployment of applications on end devices, while ensuring their secure execution on a trustworthy WebAssembly runtime. For the latter we developed Twine, whose latest revision and complete description is provided in Section 2.1. The section provides details about Twine's architecture, about its implementation, and about its evaluation. It also shows that the intended security requirements are fulfilled.

While Twine is primarily designed to serve as a platform for edge and client systems, on which SGX enclaves can be exploited, Section 2.2 provides another solution, Fortress, which is a robust and comprehensive framework that exploits ARM's TrustZone to secure IoT devices and the data they produce.

Finally, Section 2.3 describes the work towards developing a secure MQTT broker, which can be executed inside a TEE (in this case, using Intel's SGX), while also implementing TLS channels directly from the enclave to client devices. The section also refers to the deployment of the solution using Mosquitto, a well-known and open-source MQTT message broker, and its application in one of the VEDLIoT demonstrations, which also integrates with SIRE.

## Task 5.3: Simulation platform for development of security features and robustness testing

The main goal of this task was to provide a platform to project participants and future users of VEDLIoT, which can be used as a testing and simulation solution, speeding up the development process by providing a reliable and deterministic testing environment for accelerated, FPGA-oriented ML workloads, as described in Chapter 6.

This platform was developed on top of the Renode Framework [13], an open-source simulation environment developed by Antmicro, capable of simulating complex multi-node, heterogeneous, interconnected IoT systems.

In this deliverable, an overview of Renode is given in Section 6.1, which provides the context for the subsequent sections, in which the latest improvements of the testing and simulation platform are described. In concrete, new features for automatic platform generation were added, which are described in Section 6.2. Furthermore, Section 6.3 describes the developments made to provide support to RISC-V Custom Function Units (CFUs), which is done through Renode's co-simulation integration layer. Finally, improvements to these co-simulation capabilities of Renode are described in Section 6.4.

## Task 5.4: Continuous integration workflow for verification of security and robustness

In close connection to the work done in Task 5.3, the work in Task 5.4 was meant to focus on a Continuous Integration workflow to facilitate simulation and testing. In Section 6.5 we describe the examples that were added to the `renode-verilator-integration` [16] and `renode-dpi-examples` [15] repositories, which can be used by developers to more easily create their own co-simulation scenarios, either simulating CFU units [8] or bus-connected peripherals with RTL code compiled with Verilator [183].

## Task 5.5: Safety and Robustness

This task was defined to address the safety and robustness requirements of ML-based IoT systems. Giving continuity to work previously done in this task, we present in Chapter 7 two algorithms that aim at increasing the robustness of Deep Neural Networks (DNNs), thus contributing to enabling their use in safety-critical applications, such as image processing in autonomous driving systems. Additionally, we designed a framework to handle diverse driving scenarios. It is important to mention that we not only consider problems like noise or other accidental perturbations affecting images but also adversarial attacks, intending to mislead the classifiers and force them to produce misleading predictive results during inference. Finally, we discuss in Chapter 8 the possibility of sharing data between vehicles in a traffic scenario to increase road safety. Although the availability of more data, which is received from surrounding vehicles, can clearly be useful to improve perception functions, it also raises some problems, namely the trustworthiness of the received data. This is why some of the work done in WP5, namely concerning secure communication, will be valuable in this setting.

# 2 Secure Communication and Execution on IoT platforms

## 2.1 Trusted Communication with WebAssembly and Intel SGX

We followed up on our previous work regarding the integration of Twine, a trustworthy WebAssembly runtime tailored to secure the execution of WebAssembly applications within the execution context of Intel SGX [135, 125, 144].

### 2.1.1 Twine Design

Our objective with this revision is to create a runtime for running legacy WebAssembly modules inside a secure enclave, thus enabling a two-way sandbox that ensures typical TEE security features while also mitigating and controlling data leaks, even from malicious TEE service providers, as well as providing additional trustworthy guarantees.

#### 2.1.1.1 Requirements Elicitation

We identify five requirements that guide this extended Twine's design.

**R1**: *Must support the execution of legacy software services inside the two-way sandbox*. Twine should ensure transparent execution of services whose source code can be compiled to WebAssembly.

**R2**: *Must provide communication support inside the two-way sandbox*. Wasm modules running inside secure enclaves should be able to communicate through selected application-level protocols or secured socket API.

**R3**: *Must provide file management support inside the two-way sandbox*. Wasm modules running inside secure enclaves should be able to perform restricted file system operations based on runtime-defined permissions or access paths.

**R4**: *Must prevent covert channels through the available OS services*. Twine should protect the exfiltration of sensitive data from inside the enclave, which may occur through OS service interfaces.

**R5**: *Must be verifiable*. Twine should provide support for verifying the integrity of the runtime and validating its related security policies.

#### 2.1.1.2 Architectural Overview

Twine comprises two main building blocks: a Wasm runtime and a WASI interface (see Figure 2.1). The Wasm runtime runs entirely inside the TEE, and WASI works as a bridge between trusted and untrusted environments, abstracting the machinery dedicated to communicate with the TEE facilities and the underlying OS. The TEE-hosted Wasm runtime uses the WASI interface, which is always involved whenever OS services are accessed, allowing it to filter security-sensitive OCALLs. Doing so serves as an intermediate control layer preceding the actual interaction with the OS, adhering to a capability-based security approach. The runtime can limit what Wasm can do on a program-by-program basis, preventing Wasm code from leveraging the full access rights of the user owning the process. For instance, WASI may restrict an application to only open a file system subtree, similar to the capabilities of *chroot*.

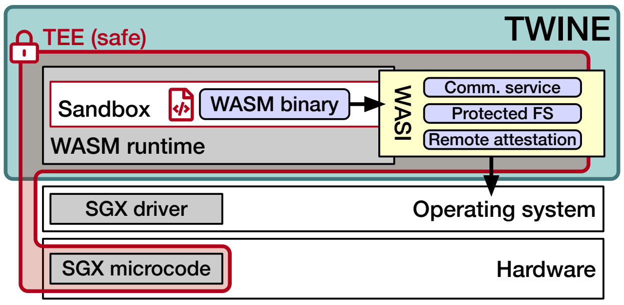


*Figure 2.1. Overall* Twine *architecture.*

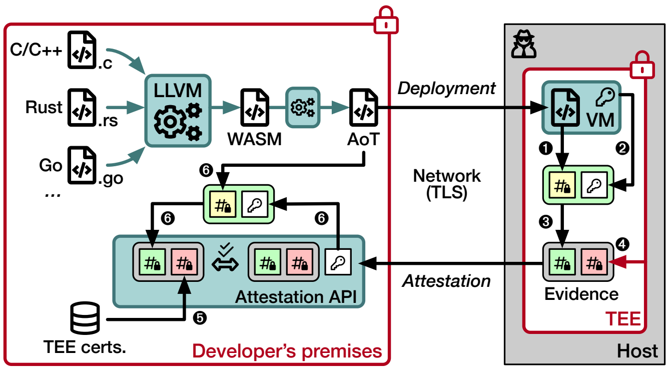


*Figure 2.2. Overview of* Twine*'s deployment and attestation workflow.*

The combination of the sandbox capabilities of SGX and WASI ends up in a two-way sandboxing system, partially inspired by MiniBox [109], and somehow follows the same perspective of Ryoan. The system, considered untrusted in the threat model of SGX, cannot compromise the integrity of the enclave code nor the confidentiality of the data stored in memory. Likewise, Wasm applications, considered untrusted from the data owner's standpoint, cannot interact directly with the OS unless WASI explicitly grants permission in the Wasm runtime.

Figure 2.2 depicts the workflow of our proposal. We compile source code from potentially multiple programming languages to Wasm bytecode and then perform AOT compilation to native code for enhanced performance. We deploy these AOT-compiled applications via a secure communication channel to ensure code confidentiality and integrity. Details on the attestation process for the hosted application and the runtime are elaborated in §2.1.2.5.

**Compliance with R1.** Twine facilitates the porting of legacy applications. In fact, it is language-independent. The programming language can be freely chosen, provided it can be compiled with LLVM or another compiler that supports Wasm and WASI as a compilation target. This lifts the restrictions imposed by SGX, typically forcing enclaved applications to be written in C/C++. Furthermore, it is cross-platform hardware-compatible. Applications can be safely executed as long as the TEE is able to execute Wasm (supported by WASI), opening the door to other TEE technologies. Finally, it is system-agnostic as long as the OS can provide an equivalent of the API required by WASI. Since WASI mimics the system calls of POSIX systems, many Unix-like variants can implement it.

**Compliance with R2, R4.** We have appropriately instrumented the WASI interface and the runtime to control security-sensitive functionalities, which could be exploited to exfiltrate data. Depending on the selected configuration, low-level communication capabilities (socket-related OS calls) may not be provided directly to the Wasm binary. In such a case, communication over the network would only go through application-level protocols (*i.e.*, HTTP) to prevent covert channels via network system call interfaces. To this end, we embedded an HTTP library inside the Wasm module, which can be configured in terms of whitelisted targeted endpoints.

**Compliance with R3, R4.** At the same time, file management support is provided and shielded against data leaks from code running in the Wasm module. The WASI interface has been extended so that every file operation leverages the Intel protected file system (IPFS), which automatically encrypts data coming out of the two-way sandbox (*e.g.*, via `fwrite`) and decrypts data flowing in the other direction (*e.g.*, via `fread`). Furthermore, the WASI sandbox enforces limitations on the Wasm applications by restricting them to predefined file system operations and access paths.

**Compliance with R5.** We also equipped Twine with an attestation service, which empowers data owners by enabling them to verify the authenticity and integrity of both the runtime environment and the WASI interfaces through which their data interacts. By implementing attestation mechanisms, data owners can confidently ensure that the runtime has not been tampered with and that the interfaces adhere to the specified security policies. This proactive approach to attestation fosters a foundation of trust between data owners and the runtime, assuring them that their sensitive information remains protected in the Wasm code and that their interactions with the runtime occur within a secure and verifiable environment, ultimately enhancing the overall security posture of the system. We equipped the Wasm module with an attestation library, which allows the data owner to generate an attestation quote whose *EnclaveHeldData* field embeds the runtime hash, the security policy hash, and the wasm module hash.

By default, Intel SGX ensures the integrity of the enclave binary rather than its confidentiality. While integrity is verified through a signature of the code, the code itself must remain in plaintext to be loaded into enclave memory. Extensions, such as Intel SGX protected code loader (PCL), do provide confidentiality guarantees for enclave binaries, albeit with some security considerations, including writable sections and segments. These considerations may result in editable enclave code and read-only data at enclave runtime [92]. Conversely, Twine can also offer code confidentiality for enclave binaries. Wasm code can be either downloaded into the enclave following attestation or retrieved from a sealed blob and subsequently loaded by Twine. Upon decryption of the Wasm code, it is mapped into a secure memory area of SGX known as *reserved memory*. This memory region allows the runtime to load arbitrary executable code, and due to the inherent robustness of Wasm, such code remains immutable from the perspective of the Wasm application.

### 2.1.2 Implementation Details

#### 2.1.2.1 Wasm Runtime and WASI

We considered many Wasm runtimes as candidates for implementing Twine. We have chosen WAMR for its small size, few dependencies, and its ability to be linked to binary code (albeit generated ahead of time, that is, no JIT). A small TCB reduces the attack surface of the runtime. Further, AOT-compiled Wasm applications achieve near-native execution speed, as shown in §2.1.4. As such, we forked WAMR and extended its WASI interface, as explained below, in such a way that we can abstract the enclave constraints while implementing systems calls.

WASI is the interface through which Wasm applications communicate with the outside world, similar to POSIX's capabilities for regular native programs. The development of TEE-enabled applications requires to deal with crossing the boundary between trusted and untrusted environments, materialised with `ECALLs` and `OCALLs` in the case of Intel SGX TEEs. We believe that leveraging WASI as the communication layer meets the purpose of Wasm, where the implementation is abstracted away for the application itself. As a result, the applications compiled in Wasm with WASI do not require any change to be executed inside Intel SGX or other TEE technologies. For example, WaTZ showcased how the same Wasm applications can be hosted in TrustZone.

By the time Twine was developed, WAMR already included a WASI implementation that relies heavily on POSIX calls. POSIX is not available inside SGX enclaves, so the WASI layer written by the authors of WAMR needs to cross the trusted boundary of the enclave frequently and straightforwardly routes most of the WASI functions to their POSIX equivalent using `OCALLs`. While this approach enables running any Wasm applications that comply with WASI inside an enclave, it does not bring additional security benefits regarding the data that transits through POSIX, as there is no encryption.

We designed Twine to implement a more optimised WASI interface for WAMR, better tailored to SGX enclaves, which adopts a different approach than plain forwarding WASI calls outside the enclave. The rationale for this choice is as follows. First, performance: most WASI calls would simply be translated to (costly) `OCALLs`. Second, we wanted to leverage trusted implementations when available, for instance, Intel protected file system (IPFS), described below (§2.1.2.4). Therefore, we refactored WAMR's WASI implementation to keep its sandboxing enforcement, splitting the remaining into two distinct layers: *(i)* one for specific implementations when available and *(ii)* and another for generic calls. Generic calls are handled by calling the POSIX library outside the enclave while providing additional security measures and validity checks. Such calls are only wired when no trusted compatible implementation exists. For instance, time retrieval is not supported by Intel SGX. Hence, Twine's WASI layer fetches monotonic time while ensuring that the returned values are always greater than the previous ones. If, for a given function, a corresponding trusted implementation exists (as it is the case for the ones in the official Intel SGX SDK), we use it to handle its related WASI call. Often, a trusted implementation calls outside the enclave while at the same time providing additional security guarantees. One notable example is the protected file system (see §2.1.2.4). Finally, Twine can disable untrusted POSIX implementations inside the enclave (via a compilation flag). This is useful when one requires a strict and restricted environment or assesses how the applications rely on external resources. In particular, the WASI interface may expose states from the TEE to the outside by leaking sensitive metadata in host calls, *e.g.*, usage patterns and arguments, despite the returned values being checked once retrieved in the enclave.

In its current implementation, Twine requires exposing a single `ECALL` to supply the Wasm application as an argument. This function starts the Wasm runtime and executes the start routine of the Wasm application, as defined by WASI ABI specifications [190]. Twine is versatile and can be adapted to only receive the Wasm applications from trusted endpoints supplied by the applications providers, as shown in Figure 2.2. The endpoint may either be hard-coded into the enclave code and, therefore, part of the SGX measurement mechanism that prevents binary tampering, or provided in a manifest file with the enclave. The endpoint can verify that the code running in the enclave (*i.e.*, the runtime) is trusted using SGX's remote attestation. We propose a remote attestation API for Wasm applications in Section 2.1.2.5. As a result, Twine is a secure deployment and execution framework for running applications on untrusted devices and environments. Despite OS dependency for network communication, Twine provides cryptographic techniques to create TLS and HTTPS channels that cannot be eavesdropped on. For that purpose, we compiled and integrated a Wasm cryptographic library in the runtime (see §2.1.2.3).

#### 2.1.2.2 Memory allocation

Memory management greatly impacts the performance of the code executed in enclaves. WAMR provides three modes to manage the memory for Wasm applications: *(1)* the default memory allocator of the system, *(2)* a custom memory allocator, and *(3)* a buffer of memory. We found that using the SGX memory allocator to enlarge the linear memory of the Wasm runtime performed poorly, leading to a time complexity above linear. Consequently, Twine preallocates a buffer of a fixed size to operate. This approach is generally not problematic, as it requires specifying a fixed heap size at compile-time for SGX enclaves.

We note that the newer iterations of Intel SGX include a memory allocation scheme called enclave dynamic memory management (EDMM) [121], enabling more memory allocation than the size specified at build time. In such cases, Twine can leverage this new capability to extend the preallocated buffer dynamically and seamlessly.

#### 2.1.2.3 Communication Support

Bringing network capabilities inside enclaves is essential to support communication with external endpoints, such as trusted peers or other enclaves. Therefore, we implemented the required calls to deal with network sockets in our WASI layer, relying on the network stack of the untrusted OS. We perform our cryptography inside the enclave to secure the communication channels and prevent attackers from eavesdropping. For that purpose, we use WolfSSL [58], an open-source popular cryptographic library. WolfSSL supports mainstream ciphers and the TLS protocol, which can be used to set up trusted communication channels. Using a renowned cryptographic library compiled in Wasm has many advantages: *(1)* the library is platform-independent and reusable in other TEEs (*e.g.*, TrustZone with WaTZ [134]), *(2)* the library can be statically linked to any application when compiled into the Wasm format, and *(3)* the library can also leverage the multi-module feature of WAMR, the runtime which Twine is based on, which enables Wasm applications to load dependencies dynamically (*i.e.*, at runtime), which eliminates the burden of static linking, and abstracts a specific implementation of the library. A WASI proposal already exists to bring cryptography to Wasm applications by the runtime. However, we considered its status too preliminary to be considered as a building block.

Furthermore, we adapted Mongoose [57], a lightweight and embeddable Web server library, to enable compilation into Wasm and facilitate the hosting of Web applications within the TEE. In conjunction with WolfSSL, our adaptation of Mongoose enables clients to establish secure communication channels featuring HTTPS termination secured by Intel SGX. Consequently, enclaved applications can expose high-level APIs, such as REST, while ensuring the confidentiality of data and the integrity of executing code. The literature has examined various approaches to providing attestation in conjunction with high-level APIs supported by TLS. Intel proposed an X.509 certificate extension, introducing specific object identifiers (*i.e.*, fields) to bind TLS and attestation [186]. Other researchers have extended the TLS handshake to incorporate additional attestation information [23]. Twine can employ one of these solutions for TEE attestation.

The WASI calls related to the sockets are implemented using `OCALLs`, forwarding them to the untrusted OS. Analogous to the WASI implementation for Linux, our approach supports the sandboxing of networking, allowing the enclave launcher to supply IP ranges to which Wasm applications can connect. We minimised the number of `OCALLs` by embedding computations that do not require untrusted OS system calls, *e.g.*, text to binary IP address conversions. WolfSSL has been slightly adapted to be compiled in Wasm. Our work builds upon the compilation target of WolfSSL for Intel SGX, with missing dependencies addressed using WASI calls and WASI-SDK header files. Due to the constraints of the Wasm virtual machine, which prohibits embedding assembly instructions in the bytecode, we could not use the hardware acceleration offered by modern CPUs for specific ciphers (*e.g.*, AES). Limitations can be mitigated by offloading certain cryptographic operations to the runtime. Mongoose exclusively supports OpenSSL and Mbed TLS libraries for providing cryptographic primitives. As such, we integrated WolfSSL as a TLS provider within Mongoose, enabling it to host or query HTTPS-enabled websites. We plan to contribute these changes to WolfSSL's and Mongoose's repositories.

#### 2.1.2.4 File Management Support

As a showcase of the abstraction power offered by WASI, we implemented the subset of the WASI calls related to file system operations by using the Intel protected file system (IPFS) [90]. Being shipped with the Intel SGX SDK, it mimics the POSIX functions for file input/output. The architecture of IPFS is split in two: *(1)* the trusted library, running in the enclave that offers a POSIX-like API for file management, and *(2)* the untrusted library, an adapter layer to interact with the POSIX functions outside of the enclave, that actually read and write on the file system. Upon a `write`, content is encrypted seamlessly by the trusted library before being written on the media storage from the untrusted library. Conversely, content is verified for integrity by the enclave during `read`.

IPFS uses AES-GCM for authenticated encryption, leveraging the CPU's hardware acceleration. An encrypted file is structured as a Merkle tree with nodes of a fixed size of 4 KiB. Each node contains the encryption key and tag for its children nodes. Thus, IPFS iteratively decrypts parts of the tree as the program in the enclave requests data [173]. This mechanism ensures the confidentiality and integrity of the data stored at rest on the untrusted file system. While the enclave is running, the confidentiality and the integrity of the data are also guaranteed by SGX's memory shielding.

IPFS has several limitations, which are deemed beyond the scope of Intel's threat model. Since the files are saved in the regular file system, there is no protection against malicious file deletion and swapping. Consequently, IPFS lacks protection against: *(1)* rollback attacks: IPFS cannot detect whether the latest version of the file is opened or has been swapped by an older version, and *(2)* side-channel attacks:

IPFS leaks file usage patterns and various metadata such as the file size (up to 4 KiB granularity), access time and file name. We note how Obliviate [2], a file system for SGX, partially mitigates such attacks. Although Obliviate can be adapted for use with Twine, addressing side-channel attacks falls beyond our threat model.

The WASI API includes several calls that do not have direct counterparts in the IPFS, due to slight variations from the ISO C standard. For instance, `fseek` allows the cursor to move past the end of a file, which is not permitted in IPFS. To address this, our WASI implementation extends the file with `null` bytes, requiring extra IPFS calls. Also, IPFS lacks support for vectored read and write operations. Since WASI exclusively handles file I/O with vectored operations, we resolved to implement those with an iteration.

IPFS provides convenient support to automatically create keys for encrypting files, derived from the enclave signature and the processor's (secret) keys. While automatic key generation seems straightforward, a key generated by a specific enclave in a given processor cannot be regenerated elsewhere. IPFS circumvents this limitation with a non-standard file open function, where the caller passes the key as a parameter. Our prototype relies on automatic generation, and we leave it as future work to extend the SGX-enabled WASI layer to support custom encryption keys.

In conclusion, files persisted by Twine cannot be read outside the enclaves and are transparently decrypted and integrity-checked while handled by Wasm applications.

#### 2.1.2.5 Attestation

RA is a cornerstone feature of TEEs, as it ensures the authenticity of the executing code, including Wasm applications in the context of Twine. We worked with the open-source community of WAMR [132, 201] to define additional functions in the runtime to interact with the attestation features of Intel SGX. Consequently, Wasm applications within our system can interface with Intel SGX to generate quotes during attestation. A quote refers to a signed data structure that contains the enclave's measurement, identity, and additional metadata, certifying the enclave's trustworthiness, integrity, and authenticity to remote parties or Wasm applications executing in other SGX enclaves. The integration of attestation within the runtime provides robust security guarantees for remote peers, typically facilitating the establishment of secure communication channels to transfer confidential data in remotely executed Wasm applications.

The RA feature of the Wasm runtime is implemented using `librats`, a low-level library facilitating attestation for multiple TEEs [88]. Although the current implementation supports Intel SGX data centre attestation primitives (DCAP), the runtime can be extended to incorporate additional TEEs. Figure 2.2 illustrates the attestation workflow. A hash is computed upon loading the Wasm application bytecode (or the AOT-compiled assembly code) within the enclave (❶). When collecting a quote, the runtime retrieves this precomputed hash and derives a secondary hash, incorporating optional user data provided by the Wasm application, such as a public key to establish a trusted communication channel (❷–❸). The runtime then forwards the final value to the Intel SGX RA mechanism, which generates a quote (❹). This quote serves as evidence for a relying party, confirming the enclave's genuineness (❺) and the trustworthiness of the executing Wasm application (❻).

### 2.1.3 Use Case: Credit Scoring in Crypto Finance

The Twine runtime has been used in a data-intensive, large-scale commercial application to enhance the trustworthiness of a distributed credit scoring oracle. *Credit scoring* has been historically used by financial institutions to estimate the risk of lending money to an individual. It determines the ability of an entity to repay debts based on a number of quantitative and qualitative metrics. High credit scores lead to higher chances of obtaining a loan with low interest rates. Conversely, entities with a lower credit score must pay higher interest rates on their loans. The *Credora Inc.* company provides a privacy-preserving scoring solution for credit in crypto-currency finance. With over \$$100$B of crypto-collateral being used to generate over \$$1.25$B of interest on a quarterly basis, credit is one of the most rapidly growing sectors of the emerging crypto-currency finance ecosystem. *Credora* allows borrowers to supply lenders with real-time portfolio risk metrics, while preserving the privacy of trades, positions, and other sensitive information. Borrowers benefit from improved lending terms, as they can display their risk in real-time and assure lenders they are trading responsibly. Lenders benefit from increased visibility and real-time information. *Credora* calculates various dynamic risk metrics and aggregates across client portfolios, including total assets, liabilities and maximum loss simulations. The latter is based on the standard portfolio analysis of risk (SPAN) system, wherein the worst possible loss of a client's portfolio is estimated from a simulation over price and volatility shock scenarios. The fundamental requirement of the *Credora* solution is that users' confidential data remains private and is computed correctly, *i.e.*, metrics must reflect the actual status of the credit. To this end, *Credora* uses TEE technologies and cryptographic proofs to ensure users' sensitive data privacy and guarantee risk analysis. Intel SGX is the enabling TEE technology adopted to protect and attest sensitive processing. Within the threat model supported by Intel SGX, only computations authorised by the user are allowed in the attested enclave, and no party can see granular private data or perform any knowledge extraction.

Figure 2.3 shows the architecture of the credit scoring system of *Credora*. It is composed of a set of distributed and loosely-coupled microservices communicating via a distributed in-memory data store. The typical execution flow is as follows. Initially, before interacting with the backend, the client challenges the KMS for its attestation (❶) to set up a secure channel. Once completed, it can provide secrets (*i.e.*, *exchange keys*) to the *Credora* TEE-secured KMS (❷). The *dispatcher* defines pulling jobs, *i.e.*, a request to be executed for a particular exchange using its API endpoints that returns information on clients' portfolios. These jobs are transferred through a shared cache to the pullers (❸). Based on those, the *private puller* uses the *exchange keys* distributed by the KMS (❹) to get clients' data (*e.g.*, information on open trading positions) from crypto-currency exchanges venues (❺) such as *Binance*, *Deribit*, *Coinbase* and *Kraken*. A trading position denotes an individual's or entity's ownership stake in a particular financial asset, which represents their investment and potential for profit or loss. The obtained data is encrypted and pushed into the shared cache (❻). Similarly, the *public puller* gathers public market data (❺), which is also stored in the shared cache (❻). The *aggregator* reads clients' private data from the shared cache, markets public data obtained by the *public puller* (❼), and computes the risk metrics and credit scores (❽).

In this use-case, the confidential data is given by: *(1) exchange keys* used to obtain clients' wallet data from *exchange* venues, and *(2)* the client trading positions received

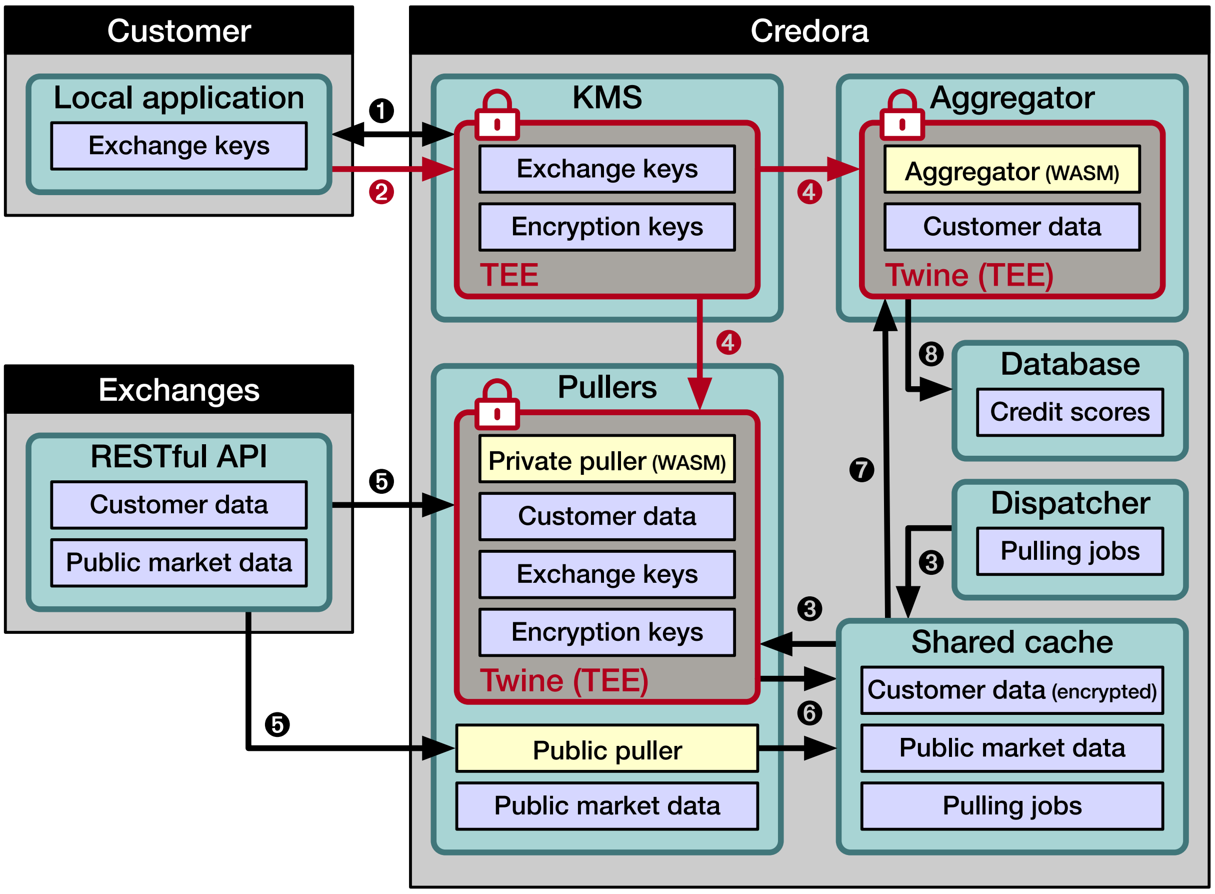


*Figure 2.3. Architectural overview and workflow of* Credora*, highlighting attested channels and* Twine *enclaves in red.*

from *exchanges*. This architecture guarantees that confidential data never leaves the secure enclave unencrypted. The *exchange keys* are received directly from clients' browsers over an attested TLS TEE-terminated secure channel. This is possible through the *librats* library [88], which can be compiled to Wasm with *emscripten* and executed inside the browser. These *exchange keys* are persistently stored using IPFS that encrypts the secret information in the enclave using the SGX sealing key. The process of signing requests for *exchanges* using the *exchange keys* is executed inside the enclave. RESTful requests for *exchanges* are then made over an HTTPS TEE-terminated connection using WolfSSL and Mongoose compiled in Wasm, guaranteeing that the confidential financial data is directly received and aggregated in the enclave.

Under the described scenario, *Credora*'s clients only need to trust Intel (*i.e.*, SGX threat model) and *Credora* itself, which does not disclose the implementation of their software. Hence, the cloud provider hosting the application may be untrusted, thanks to the isolation offered by Intel SGX. Additionally, we aim to further reduce the scope of the threat model by removing *Credora*, leaving Intel as the only trusted entity. Toward this goal, one must prove to clients that no data has ever leaked from the secure enclaves, nor that the enclaves are curious and retrieve unwanted information from the cloud provider's system. As such, Twine's trust model plays an essential role: its two-way sandbox (as explained in §2.1.2) offers two strong guarantees for the cloud provider and the clients. First, the *Credora*'s enclaves, which are based on Twine, are proven authentic using RA, which guarantees to *Credora* that the enclaves have not been tampered with and can supply Wasm applications and confidential information for further computations, preventing one from eavesdropping. Second, *Credora*'s customers can review the open-source implementation of *Credora*'s enclaves (*i.e.*, based on Twine), ensuring that the runtime properly sandboxes the Wasm applications (whose source code is proprietary) deployed by *Credora* later on, which prevents the *Credora*'s Wasm applications from accessing the host system resources.

### 2.1.4 Evaluation

We present here the new benchmark results from our evaluation of the extended version of the runtime Twine. For the full set of our previous benchmarks, refer to Deliverable 5.1 from VEDLIoT [144]. We intend to answer the following questions:

- What are the performance overheads for using cryptographic operations and setting up TLS-terminated connections within the enclaves?
- How does Twine perform when used in a data-intensive and real-world solution?

We answer these questions by encrypting, hashing and securing communications using cryptographic primitives and TLS with WolfSSL (§2.1.4.2), and assessing the end-to-end performance of a Wasm application in the fintech company *Credora* (§2.1.4.3).

#### 2.1.4.1 Experimental setup

We use a Supermicro 5019S-M2 with Intel Xeon E3-1275 v6 (3.8 GHz, EPC 128 MiB, usable 93 MiB) for SGX tests. Systems run Ubuntu 18.04.6 with kernel 4.15.0-202, SGX driver v2.11, and SGX SDK v2.17.100.3. Twine is fully merged into the WAMR's official repository. As such, we use the WAMR build for running the benchmarks unless stated otherwise.

Time is measured using the monotonic clock of POSIX in all the benchmarks and averaged using the median. If measured from within the enclave, the time to leave and reenter the enclave is included. In our setup, the enclave round trip accounts for approximately 4 ms. We used Docker to build the benchmarks while their execution is on bare metal to avoid potential isolation overheads. The native benchmarks are compiled using Clang 10 with optimisation set to `--O3`. The Wasm benchmarks are compiled using Clang into Wasm format, then AoT-compiled into native format using the compiler provided by WAMR (*i.e.*, `wamrc`) using `--O3` and size level 1 to run into SGX enclaves (`--sgx`). We used GCC v7.5 for two tasks: *(1)* compile the applications executing the benchmarks, *i.e.*, the WAMR runtime and the SGX enclaves, also with `--O3`, and *(2)* compile IPFS with `--O2`, as in the SGX SDK. SGX-LKL (v0.2.0), LKL (v5.4.62) and AccTEE have been used as an empirical baseline for running the experiments natively in SGX enclaves. They have been downloaded from the official Debian repository and GitHub. Finally, our implementation and instructions to reproduce our experiments are open-source [133].

#### 2.1.4.2 Micro-benchmarks: network stack

We assess the network stack of Twine using two micro-benchmarks bundled with WolfSSL: *(1)* a performance comparison of cryptographic primitives using many ciphers and hashing algorithms, and *(2)* a performance evaluation of TLS sessions, which have one side of the connection terminated within the enclave. In both cases, we compare the relative execution speed of native and Wasm (inside SGX) and Wasm (outside SGX) against native (outside SGX). For a fair comparison between native and Wasm, we have chosen to disable the hardware acceleration support of WolfSSL, *i.e.*, the offloading of some ciphers using CPU capabilities.

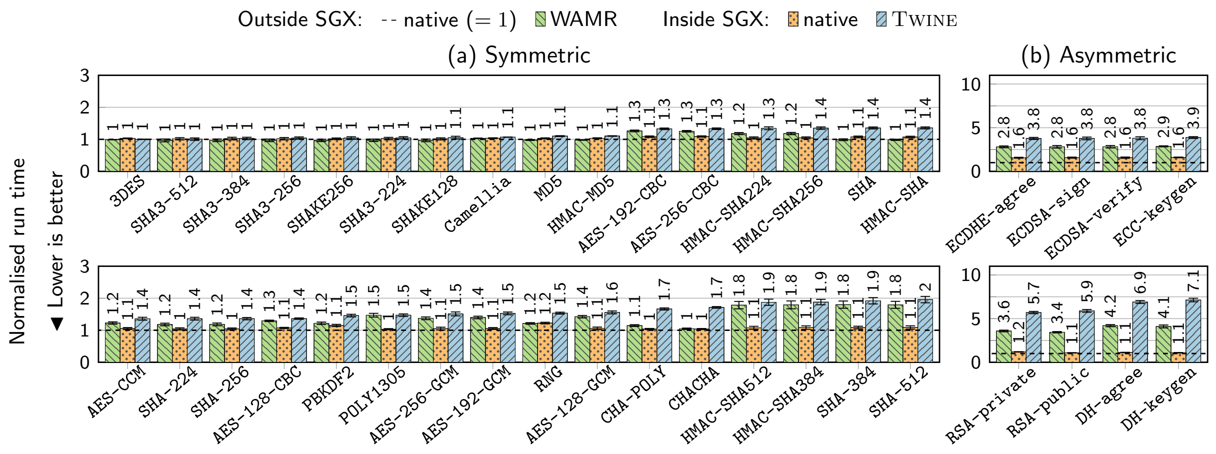


*Figure 2.4. Performance of WolfSSL benchmarks targeting cryptographic algorithms, normalised to the native speed.*

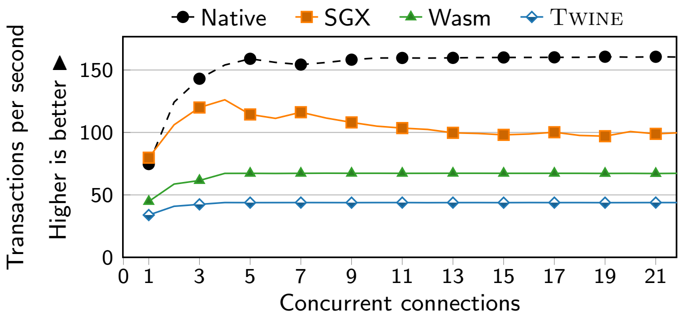


*Figure 2.5. Performance of WolfSSL benchmarks for TLS sessions.*

Figure 2.4a depicts the execution speed of symmetric algorithms and hashing functions. In contrast, Figure 2.4b illustrates the speed for asymmetric ciphers and key generation operations. Among these, symmetric operations are the most efficient, with an average slowdown for Wasm of 1.2× outside of SGX, and 1.3× inside of the TEE, compared to native speed execution. Hashing functions follow, with slowdowns of 1.3× and 1.4× for Wasm outside and inside SGX, respectively. Finally, the asymmetric ciphers have the highest slowdown, with 3.3× and 5.1× for the same settings. The TLS protocol uses asymmetric ciphers to establish sessions, authenticate remote endpoints, and exchange session keys. Once the session is active, it relies on more efficient symmetric and hashing algorithms, mitigating concerns over the slower speed of asymmetric ciphers.

In Figure 2.5, we stressed an application using WolfSSL's TLS 1.3 protocol by evaluating the number of TLS transactions per second over a range of concurrent connections. The setup involves a client and a server hosted on different machines connected through a switch. The client is a native executable running on Linux, while the server is of four types and evaluated separately: native in Linux and SGX, Wasm in Linux and Twine in SGX. A TLS transaction is comprised of *(1)* the TLS handshake, *(2)* the server reading 16 KiB, *(3)* the server sending 16 KiB, and *(4)* the closure of the session. We measure the time the server takes to handle 512 connections while varying the number of concurrent connections handled by a single core. Besides, we used the ciphersuite `TLS13-AES128-GCM-SHA256` as cryptographic primitives. Finally, we considered `AES128-CCM` and `CHACHA20-POLY1305` as AEAD ciphers, but we did not notice significantly different results because the performance of these algorithms is similar, as reflected in Figure 2.4.

The number of transactions per second for native execution outside of SGX and Wasm demonstrates a pronounced increase until it reaches a maximum value, after which it stabilises. This maximum value signifies the saturation of the singular thread responsible for managing TLS sessions. The native execution within the enclave exhibits a comparable pattern; however, it necessitates a more extended period to converge after attaining its peak value. Although we did not conduct a comprehensive investigation on this particular behaviour, we believe the observed phenomenon results from how SGX-LKL (the library operating system employed for the execution of native applications in SGX) processes in-enclave packets via its dedicated TCP/IP stack. Native execution outside of SGX converges to an average of 157 TPS, which serves as the baseline measurement. In contrast, native execution within the TEE exhibits an average of 97 TPS with a consequent slowdown factor of 1.6$\times$. Furthermore, the average transaction rates per second for Wasm outside and inside the enclave are 67 TPS and 44 TPS, respectively, with corresponding slowdown factors of 2.4$\times$ and 3.6$\times$. When comparing the in-enclave solutions, Twine's slowdown relative to native is 2.2$\times$, but offers all the advantages of Wasm, such as portability and security. Yet, Wasm services hosting TLS connections may leverage multithreading for performance enhancement.

We highlight that modifications were made to the official benchmark to use the `poll` system call instead of `epoll`, as the latter is unsupported in WASI. The resulting software was then contributed to the WolfSSL open-source repository.

#### 2.1.4.3 Macro-benchmark: Twine for Credit Scoring

We estimate the impact of Twine on the credit scoring application (see §2.1.3). Our goal is to evaluate the overhead of Twine for time-sensitive functionalities impacting business operations. In particular, we focused on the *Private Puller* and *Aggregator* components, measuring the duration required to pull clients' data from Exchanges and to compute portfolio aggregations, respectively. In both instances, performance is critical, as producing outdated data may impair data accuracy and subsequently undermine *Credora*'s credibility.

The initial experiment entailed measuring the time to complete each request from single-thread clients. We target a variety of Exchange venues by querying their API endpoints which return JSON data with a maximum size of 8 KiB, containing clients' positions (*i.e.*, an individual's ownership in a financial asset, reflecting their investment and potential for gains or losses). The baseline relies on an extended version of the `cpp-httplib` library [54], which uses SGX-WolfSSL [59] to carry out TLS cryptographic functions, adopted by *Credora* in production. On the other hand, Twine uses WolfSSL and Mongoose. Figure 2.6 reports the results of the *Private Puller* evaluation. It is apparent that the two solutions exhibit comparable behaviour. For some Exchanges, Twine is slower (*e.g.*, up to $30\%$ for Binance), while for others, it yielded better pulling time (*e.g.*, up to $17\%$ for GateIO). This suggests that the performance is similar and subject to variations attributable to the server-side processing.

Next, we measured the duration required to complete a single aggregation round of

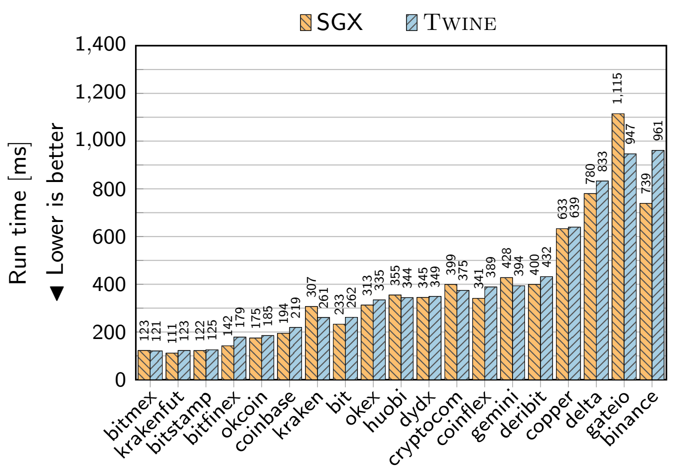


*Figure 2.6. Pulling time of* Credora Private Pullers *using SGX and* Twine.

all clients' entries (*i.e.*, a total of 522 in April 2023) available in the *Credora* production environment. Utilising nine parallel aggregators, we obtained the following results:

$$t(agg)_{without\_\text{Twine}} = 3\text{m}23\text{s}$$
$$t(agg)_{with\_\text{Twine}} = 4\text{m}51\text{s}$$

The runtime overhead using Twine is considered acceptable for *Credora*, which can effortlessly scale its computing units, albeit at the expense of increased infrastructural costs. Upon further profiling, we discovered that the primary source of delay is attributable to the JSON library managing large data chunks. *Credora* uses the *nlohmann* library [115], which is characterised by substantial memory consumption and consequently suboptimal performance when executed within Twine. In future, we plan to adopt more memory-optimised JSON libraries to mitigate this overhead further.

### 2.1.5 Security Analysis

In this section, we analyse how Twine contributes to the security of the requirements specified in §2.1.1.1. Additionally, we compare the security standpoint of Twine to some state-of-the-art solutions, namely AccTEE [76] and SGX-LKL [151].

**Security of R1.** The two-way sandbox in Twine offers security through two mechanisms: *(1)* Intel SGX, which protects against the system tampering with Twine and the Wasm application, and *(2)*, a Wasm sandbox to enforce memory safety while requiring the hosted application to rely on WASI for any interactions with the untrusted OS. These dual mechanisms allow both the application and infrastructure providers to confirm the integrity of the sandbox, which neither party cannot alter. For comparison, AccTEE also offers a two-way sandbox but lacks WASI support, thereby limiting the application's ability to access OS services and constraining the portability of legacy applications. SGX-LKL, conversely, does not offer a two-way sandbox but enables the execution of legacy applications through its ad-hoc variant of libc.

We observe that AOT-compiled code can bypass the Wasm sandbox if not compiled in a secure environment. Typically, the compilation of Wasm bytecode into assembly code ensures that memory access and function calls stay confined to the sandbox. However, malicious actors could craft assembly code that executes unauthorised operations in Twine, like directly invoking an OCALL function. To counter this, we suggest three mitigations: *(1)* use JIT compilation to ensure secure code compilation within Twine, albeit at a performance cost, *(2)* establish a separate, secure enclave solely for compilation that communicates with Twine, or *(3)* coordinate with the Wasm application and infrastructure owners to validate the hash of the AOT-compiled assembly against the loaded code in Twine, provided that both parties having prior knowledge of the Wasm bytecode. The first option is straightforward and already supported in Twine (as it is based on WAMR), while the latter two are more challenging to tackle.

**Security of R2 and R4.** In Twine, hosted applications can leverage two pre-compiled Wasm components for secure communication: a lightweight TLS library and an HTTPS library for establishing secure communication channels. These components enable the creation of TLS-termination endpoints within the enclave, ensuring both data confidentiality and integrity during transmission. Furthermore, Twine may extend Wasm's capability-based security model to permit application-level protocols exclusively like HTTPS, mitigating the risk of data exfiltration through insecure communication channels. In contrast, SGX-LKL provides encrypted channels exclusively between trusted enclaves or trusted parties using Wireguard as the VPN solution. However, this approach has limitations: both endpoints must be configured with Wireguard, and including the TCP/IP stack in the TCB contributes to its increased size. Outside this trusted network, hosted applications must provide their own TLS implementation. SGX-LKL's oblivious communication feature is likewise confined to its VPN network. On the other hand, AccTEE claims to support I/O calls but delegates encryption to the application or the underlying layer, which is SGX-LKL.

**Security of R3 and R4.** Intel protected file system is integrated with WASI for secure file operations in hosted applications, including transparent encryption and decryption. Replacing standard libc calls with IPFS functions effectively mitigates the risk of exfiltrating sensitive data through the file system. The enclave's sealing key serves as the basis for a symmetric key, decrypting a file's Merkle tree root node. This root node holds essential metadata for decrypting subsequent nodes [173]. Although the Merkle tree nodes are stored on the untrusted file system, decryption is limited to the specific enclave or its owner via the enclave- or owner-bound sealing key [89]. SGX-LKL also offers file system security through the use of virtual block devices. It creates virtual disks on an untrusted file system and uses an ad-hoc algorithm to mitigate side-channel data leaks. While Twine does not guard against side-channel attacks, its abstraction through WASI allows the secure file system to be replaced with other state-of-the-art solutions like Obliviate [2]. As for AccTEE, file system security is delegated to SGX-LKL, similar to its approach to network security.

**Security of R5.** Twine offers hosted applications the capability for attesting the Wasm bytecode in JIT mode, or assembly code in AOT mode, along with the runtime. This serves two purposes: *(1)* it lets the enclave owner validate the genuineness of the hardware and the integrity of the application, and *(2)* it provides assurance to the infrastructure owner of the correct implementation of the Twine runtime. Furthermore, the attestation process ensures that a specific configuration of Twine is in place, including disabling some system calls for the Wasm module. Turning off specific system calls enhances security by reducing the attack surface, adhering to the principle of least privilege, and easing system monitoring and auditing. By comparison, AccTEE mentions attestation but lacks an API to expose the evidence to the Wasm applications for secure communication. SGX-LKL offers remote attestation by sharing the hash of the virtual disks with trusted entities. However, this approach can be challenging, especially when hosted applications store files since the attestation measurement also reflects these files. Consequently, managing multiple virtual disks becomes necessary for maintaining known attestation measurements.

#### 2.1.6 Retrospective

The reluctance to adopt distributed architectures for sensitive applications arises from a lack of trust when outsourcing computations to remote parties. Although this issue has been extensively studied in the context of TEEs, such solutions introduce non-trivial drawbacks and constraints, including limited programming language support, restrictions on system calls, and enforced programming paradigms. In this section we proposed an approach for executing unmodified programs in WebAssembly (Wasm) — a target binary format for applications written in LLVM-supported languages, such as C, C++, Rust, Go, and Swift — within lightweight TEEs that can be easily deployed across client and edge computers. Twine is our trusted runtime that supports the execution of unmodified Wasm binaries within SGX enclaves. Wasm offers several advantages, including speed, versatility, and abstraction of the complexity associated with developing applications tailored for specific TEEs. Furthermore, we provide an adaptation layer between the standard WebAssembly system interface (WASI) used by applications and the underlying OS, translating WASI operations into equivalent native system calls or functions from secure libraries specifically designed for SGX enclaves. Consequently, trusted applications can seamlessly interact with encrypted files and secure network connections via TLS and HTTPS. Our comprehensive evaluation demonstrates performance comparable to other state-of-the-art approaches while offering robust security guarantees and full compatibility with standard Wasm applications. Finally, Twine is freely available as open-source software and has been merged into the original WAMR runtime.

## 2.2 Protecting IoT Peripherals with ARM TrustZone

The widespread adoption of IoT platforms has raised serious security and privacy concerns due to the nature of interconnected devices and the vast amounts of data generated [21]. For instance, in smart homes, large amounts of sensitive data are constantly generated through sensors and hardware peripheral devices, *e.g.*, cameras, microphones. This data is usually transmitted to untrusted cloud services and third-party providers, oftentimes with little regard to the confidentiality of the shared data. The involvement of multiple parties introduces privacy issues, as the data could be used for purposes beyond the user's knowledge or control. For example, in July 2019, more than 1000 Google Assistant recordings were involuntarily leaked [79, 50], with part of these recordings activated accidentally by users. Furthermore, privileged software like the underlying operating system (OS) or hypervisor can be compromised [107, 181], potentially leading to data breaches or unauthorized access to sensitive data.

To cope with these security threats, hardware-based security technologies that allow the creation of *trusted execution environments* (TEEs) have been developed. Popular and emerging implementations include Intel software guard extensions (SGX) [52], Intel trust domain extensions (TDX) [47], and AMD secure encrypted virtualization (SEV) [60] which target server-end environments, and ARM TrustZone [149] or RISC-V MultiZone [71], respectively for low-power ARM-based or RISC-V-based client-end

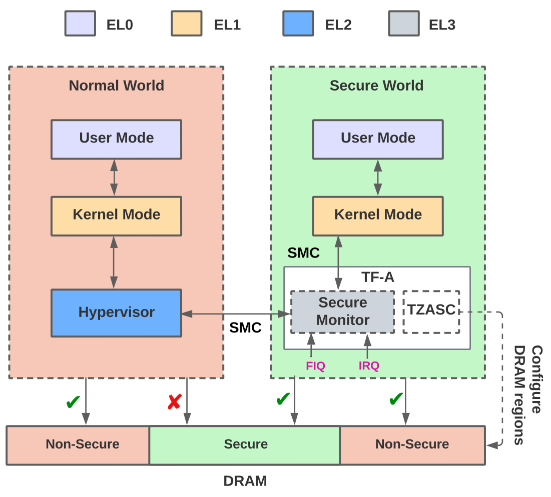


*Figure 2.7. TrustZone security architecture on ARM Cortex-A.*

devices. Although TEE frameworks like OP-TEE [179] have been leveraged to secure both user and kernel level applications, a holistic and generic solution for safeguarding peripheral data in IoT setups remains absent.

The work presented in this section aims to fill the gap described above by proposing Fortress, a robust and comprehensive framework to enhance security and privacy in IoT infrastructures. Our approach involves restricting peripheral I/O memory to a secure kernel space TEE, providing access only to a small part of peripheral driver code while isolating peripheral data from an untrusted OS or hypervisor. The data is then securely transferred to a user space TEE, where obfuscation or data sanitization techniques can be applied to encrypt or remove sensitive information before it is transmitted to an untrusted cloud environment. This security framework has practical applications in various contexts, including smart homes, medical IoT, retail IoT, amongst many others.

In summary, our research effort produced a design to secure IoT peripheral data using ARM TrustZone and a generic approach to partition peripheral drivers. These results are presented in the following sections. A scientific paper was accepted for publication [200] including our results, together with the details of a proof-of-concept implementation of our design using $I^2S$ peripherals on ARMv8-A, demonstrating the feasibility of the proposed approach, and a comprehensive evaluation of our system, which shows that privacy is achieved at a reasonable cost.

### 2.2.1 TrustZone, a TEE for Edge Devices

A *trusted execution environment* (TEE) is a hardware-enforced security technique to isolate sensitive code and data at runtime from potentially malicious entities, including privileged software like the OS or hypervisor. Various hardware security technologies have been developed that allow to create a TEE. Intel SGX [52] provides process isolation through *enclaves*, while AMD SEV [60] and more recently Intel TDX [47] focus on isolating virtual machines (VMs). These technologies are specifically tailored for server-grade environments.

For edge-based and low-power devices, ARM TrustZone (TZ) [149] is the most common TEE technology. TZ is a security technology for ARM-based processors which divides the processor into two protection domains: *secure world* wherein sensitive operations can be performed, and *normal world* for performing non-sensitive operations. Fig-

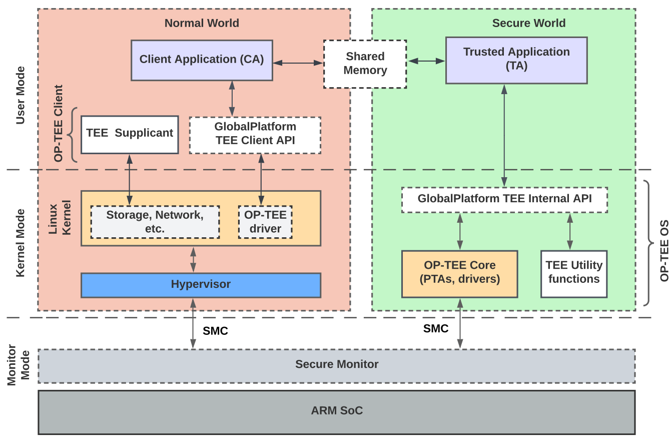


*Figure 2.8. OP-TEE architecture.*

ure 2.7 describes TrustZone's security architecture. At any point in time, the processor operates exclusively in one of these worlds. A special bit known as the *non-secure (NS)* bit stored in the *secure configuration register (SCR)* determines the current protection domain of the processor, and is used for memory access control checks across both worlds. The processor can equally transition between worlds; TrustZone introduces a new component known as the *secure monitor* (operating in *monitor mode*) which acts as a bridge between both worlds and is responsible for storing processor state during transitions. A new privileged instruction, *i.e.*, *secure monitor call (SMC)*, allows software in both worlds to switch to the opposite world via monitor mode. Regarding interrupts, IRQ (normal interrupt request) and FIQ (fast interrupt request) in the secure world can also trigger a transition to monitor mode without an SMC. The ARMv8 architecture provides four privilege levels, *i.e.*, *exception levels* (EL) [111] at which code can run: EL0 for user space code, EL1 kernel space code, *e.g.*, the OS, EL2 for the hypervisor, and EL3 for secure monitor mode. [1]

The memory infrastructure in TrustZone-enabled systems (Cortex-A) introduces a hardware component called the *TrustZone address space controller* (TZASC), which the ARM Trusted Firmware (TF-A) [112] uses to configure specific DRAM areas as secure regions. These configurations can be done such that secure world applications can access all memory regions while normal world applications are confined to non-secure memory. A similar memory partitioning functionality is performed by a component called *TrustZone memory adapter* (TZMA) but targets SRAM rather than DRAM. Both TZASC and TZMA may or may not exist on a specific system-on-chip (SoC) implementation. For example, the Raspberry Pi 3 plaftform supports some ARM TrustZone features but lacks TZASC and TZMA [165], and hence it lacks the capability of securing memory with TrustZone.

### 2.2.2 Open Portable TEE (OP-TEE)

OP-TEE [179] is a software framework that implements a trusted execution environment for TrustZone. OP-TEE is compliant with the GlobalPlatform TEE Internal/Client API specification v1.0 [73], and consists of three main components: *OP-TEE client*, *OP-TEE OS*, and *OP-TEE Linux driver*. OP-TEE client offers an API that allows applications in

[1] A prefix of "S" is usually added to the exception level for more precision when the code is being executed in the secure world. For example S-EL0/S-EL1 explicitly indicate that the user space/kernel space code is executing in the secure world.

```
tegra_i2s1: i2s@2901000 {
    compatible = "nvidia,tegra194-i2s",
            "nvidia,tegra210-i2s";
    reg = <0x2901000 0x100>;
    clocks = <&bpmp TEGRA194_CLK_I2S1>,
        <&bpmp TEGRA194_CLK_I2S1_SYNC_INPUT>;
    clock-names = "i2s", "sync_input";
    ...
    sound-name-prefix = "I2S1";
    status = "okay";
 };
```

*Figure 2.9. DT node for an I2S interface on Tegra 194 SoC.*

the normal world, also known as *client application* (CAs), to interact with applications running in the secure world, known as *trusted application* (TAs). Specifically, the CAs operate within a *rich execution environment* (REE), *i.e.*, in the regular OS at the EL0 level, while TAs execute in the secure world inside the TEE, at the S-EL0 level. OP-TEE Linux driver is a kernel-space component operating in the normal world OS (at EL1) which facilitates transitions between normal and secure worlds. In the presence of a hypervisor, the SMC from a guest kernel traps into the hypervisor and the latter performs the SMC on behalf of the former. OP-TEE OS provides an interface called a *pseudo trusted application* (PTA) which TAs can use to communicate with OP-TEE OS.

Given the large dominance of ARM-based architectures on user-end devices in IoT setups, the architecture and design of Fortress is based on ARM Trustzone and OP-TEE, since it is the dominant architecture on user-end devices in IoT platforms.

### 2.2.3 MMIO and DMA

Contemporary computer systems and IoT devices comprise peripheral devices, *e.g.*, keyboard, mouse, camera, microphone, fingerprint sensors, aimed to provide I/O capabilities. Dedicated system software called *device drivers* enable the OS kernel to interact with these peripherals. The job of a typical device driver is, for the most part, reading or writing I/O memory [51]. This is commonly achieved through mechanisms like MMIO and DMA, which we detail next.

*MMIO.* It is a method of performing I/O between the CPU and a peripheral device by mapping the peripheral's *I/O registers* into the CPU's address space. As such, processor `load`/`store` instructions on mapped addresses translate to load/store operations on the corresponding I/O registers in physical memory. The Linux kernel on 64-bit ARM platforms provides MMIO access primitives like `ioreadX`/`iowriteX` which read and write `X` bits of data from and to an I/O register, respectively. For example, `ioread8(reg)` and `iowrite8(reg,val)` read and write 8 bits of data from and to a memory-mapped register, respectively.

*DMA.* It is a mechanism used by peripheral devices to transfer I/O data to and from main memory bypassing the processor [51]. A specialised hardware component, *i.e.*, the *DMA controller*, coordinates the DMA data transfer, and generates an interrupt request to the CPU upon DMA completion.

MMIO or DMA memory for a particular peripheral device is usually associated with a *base address* and *size*, which indicate respectively the base address in physical memory, and the size of the contiguous portion of memory used by the data transfer mechanism. The peripheral's specifications provides information regarding base addresses, sizes, and the offsets for all useful registers and DMA regions. The details regarding all hardware devices on a system are described in a data structure called the *device tree* (DT). The DT contains nodes describing all the hardware on a system, including CPUs, memory, clocks, and peripheral devices like I$^2$S, Ethernet cards, *etc.* For example, Figure 2.9 represents a node in the DT of an NVIDIA Jetson AGX Xavier kit describing I$^2$S peripheral's properties, such as the base address in memory (`0x2901000`) and size (`0x100`). The DT is read by the kernel at boot time.

### 2.2.4 Threat Model

Fortress has four main security goals, aimed to guarantee the confidentiality and integrity of sensitive data (*e.g.*, images, voice recordings) originating from IoT peripherals before its transmission to the cloud:

**(G1):** Protect the confidentiality of peripheral data from the underlying OS or hypervisor.

**(G2):** Ensure a trusted path between kernel space where the data is originally obtained to secure user space where data processing (*e.g.*, encoding, decoding, filtering) occurs.

**(G3):** Ensure the confidentiality (and integrity) of sensitive peripheral data transferred to the cloud, or complete removal of sensitive data from the data stream.

**(G4):** Minimise the trusted computing base and hence the potential attack surface.

*Hardware.* We assume that the underlying hardware itself is not malicious and cannot be tampered with by the adversary. Fortress shields existing peripheral devices connected to the SoC, and does not consider the connection of new components, *i.e.*, the adversary's goal is to access sensitive data of already connected peripherals, typically via MMIO or DMA. Denial-of-service attacks (*e.g.*, maliciously shutting down the system) as well as side-channel vulnerabilities [114] are considered out-of-scope.

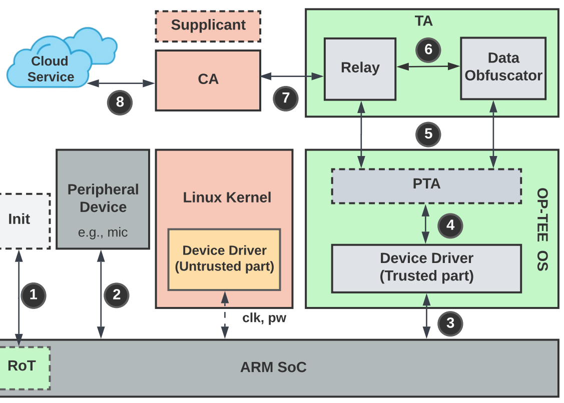
*Software.* The on-die boot ROM and intermediate firmware, as well as OP-TEE are considered trusted components, which permit establishment of a *chain-of-trust* during a secure boot process. We assume the peripheral's hardware components information, *e.g.*, MMIO or DMA physical addresses, is encapsulated in a device tree file that is signed and verified while establishing the chain-of-trust, preventing a potential attacker from misconfiguring the device. All other software on the system is considered untrusted.

*Third-party.* Any element outside the hardware perimeter of the SoC, such as the cloud providers in the IoT setup (*e.g.*, AWS IoT [30]) to which potentially sensitive peripheral data is transmitted, are considered untrusted.

### 2.2.5 Fortress Architecture

In this section, we present the architecture and design of Fortress. Here, we show how its components interact to achieve the security goals outlined in §2.2.4.

The workflow of Fortress is summarised in Figure 2.10. At system initialisation, the secure boot process authenticates system boot components, *e.g.*, OP-TEE OS image, DT files, and TZASC is used to configure peripheral MMIO and DMA memory regions



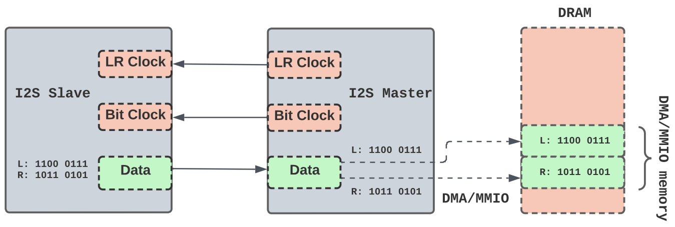


*Figure 2.11. Overview of* $I^2S$ *protocol.*

to be accessible to the secure world only (❶). After system setup, a peripheral device, *e.g.*, microphone or camera which constitutes a smart home/IoT setup, generates potentially sensitive data, *i.e.*, speech or visual data (❷). In a regular setup, the device driver software is part of the untrusted OS, thus leaking sensitive data. In Fortress, the driver is partitioned into trusted and untrusted parts, which respectively execute in the secure and normal world. The trusted part is embedded in the OP-TEE OS kernel, and has exclusive access to the generated peripheral data, which is read into secure I/O buffers (❸). The sensitive data is securely processed by the trusted driver, after which it is transferred to a trusted application via the PTA interface (❹–❺). The TA comprises a data obfuscator that acts as a firewall, encrypting or filtering out any sensitive information (❻). Converting human speech into a finite set of voice commands is a typical example of stripping potentially sensitive information. Finally, the reviewed data is sent to an untrusted cloud service via a relay module in the TA (❼–❽). The relay module leverages a Linux user-space daemon called the *TEE supplicant* to provide OS-level services such as network communication or storage.

We create a proof-of-concept implementation of this approach with secure sound processing via $I^2S$, a serial communication protocol commonly used to transmit audio data between integrated circuits (ICs). Figure 2.11 provides an overview of the $I^2S$ protocol. One IC, *i.e.*, SoC (master) controls data transfer by manipulating two clock lines: the *left-right clock* to specify the audio channel (`L=0;R=1`), and the *bit clock* to specify if data can be sent (`1`) or not (`0`). The second IC, *i.e.*, $I^2S$ microphone (slave) sends data bits (via MMIO or DMA) for the corresponding channel each time the bit clock is set.

#### 2.2.5.1 System Initialization and Peripheral Memory Isolation

*Secure boot.* To ensure the authenticity and integrity of trusted system components, ARM TrustZone supports *trusted board boot* (TBB) [22]. The latter establishes a chain-of-trust consisting of several stages of integrity verifications, starting from the boot

ROM, comprising the root-of-trust (RoT), and extending through various bootloader stages, the ARM trusted firmware (containing the secure monitor), and OP-TEE OS components. During the deployment phase, critical components of the TCB, such as OP-TEE OS, device tree files, and TAs undergo cryptographic signing (using SHA-256) with a private key. The secure boot process uses the corresponding public key to verify the signatures of these components. By verifying the integrity of OP-TEE OS, this in turn ensures the integrity of Fortress's trusted peripheral driver, as well as device tree files containing peripheral I/O registers.

*Memory isolation.* To safeguard peripheral data from unauthorised access by untrusted components like the OS and hypervisor, it is necessary to confine the memory regions associated with peripheral data to the secure world. In the context of our work, we collectively refer to these memory regions as the *secure I/O region* of the peripheral. Since most IoT hardware peripherals transfer data using MMIO or DMA mechanisms, Fortress configures peripheral MMIO and DMA memory ranges as the secure I/O region. The specific MMIO registers and DMA regions used by the peripheral are typically defined in the device's DT node. Our solution offers two methods for specifying these secure I/O regions. The first is static and involves hard-coding the physical memory addresses of the MMIO and DMA base registers (along with their sizes) directly into the driver's source code. The second approach is dynamic, where the information is read from the DT node in the corresponding device tree file.

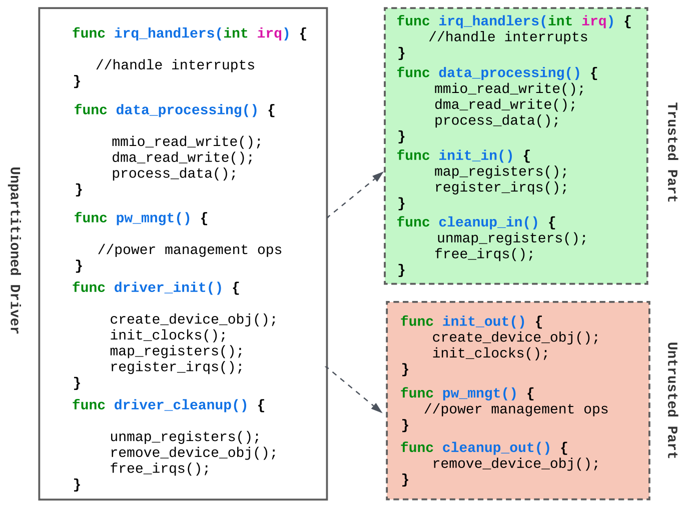
During system initialisation, the trusted firmware utilises TZASC to configure DRAM in such a way that the peripheral's secure I/O region (*i.e.*, MMIO and DMA memory ranges) is exclusively accessible to the secure world. This configuration effectively prevents any access to the peripheral's data from the untrusted OS or hypervisor, thereby satisfying the security requirement in **G1**. To complete the initialisation of the peripheral's memory, the trusted driver's initialisation code invokes OP-TEE OS's `core_mmu_add_mapping` routine, which maps the secure MMIO regions into kernel-space, ensuring that the secure driver can effectively interact with the peripheral.

#### 2.2.5.2 Secure Kernel to User-space Communication

After the trusted driver reads peripheral data via MMIO or DMA into its I/O buffers, it needs to transfer the data to a secure user-space TA for further processing operations, *e.g.*, encoding, encryption, or filtering. To establish this secure connection between the OP-TEE driver and the TA, we leverage an OP-TEE PTA. The latter acts as a bridge, offering an interface that connects the TA to secure driver space. The TA utilises GlobalPlatform Internal API to safely retrieve peripheral data from the driver's memory space via a PTA invocation. This operation requires a context switch from S-EL0 to S-EL1, and can be summarised in four steps: *(1)* checking access rights of the TA's destination buffers in the invocation, *(2)* copying necessary parameters (*e.g.*, the PTA/driver function identifier) from TA memory to OP-TEE OS memory space, *(3)* issuing a system call to OP-TEE OS to invoke the corresponding PTA/driver routine, and *(4)* copying results from OP-TEE OS memory into the TA's destination buffer. The creation of this secure channel from OP-TEE OS to the TA satisfies **G2**.

#### 2.2.5.3 Data Obfuscation

*Data obfuscation* is the process of transforming sensitive information to safeguard it from unauthorized access. Common data obfuscation techniques include encryption, randomization, anonymization *etc.* By integrating a data obfuscation module in



```
func irq_handlers(int irq) {

   //handle interrupts
}
func data_processing() {

     mmio_read_write();
     dma_read_write();
     process_data();
}
func pw_mngt() {

   //power management ops
}
func driver_init() {

     create_device_obj();
     init_clocks();
     map_registers();
     register_irqs();
}
func driver_cleanup() {

     unmap_registers();
     remove_device_obj();
     free_irqs();
}
```

```
func irq_handlers(int irq) {
    //handle interrupts
}
func data_processing() {
     mmio_read_write();
     dma_read_write();
     process_data();
}
func init_in() {
     map_registers();
     register_irqs();
}
func cleanup_in() {
     unmap_registers();
     free_irqs();
}
```

```
func init_out() {
     create_device_obj();
     init_clocks();
}
func pw_mngt() {
    //power management ops
}
func cleanup_out() {
     remove_device_obj();
}
```

*Figure 2.12. Fortress driver partitioning.*

Fortress, sensitive data originating from an IoT peripheral can be protected before its transmission to an untrusted party in the cloud or the host OS, hence satisfying **G3**. The generic architecture proposed by Fortress offers the flexibility to incorporate diverse obfuscation techniques depending on the nature of the data. Our implementation leverages techniques provided by OP-TEE's cryptography API which is based on LibTomCrypt [169], a popular open-source cryptographic library. The latter provides various symmetric cryptographic algorithms, including AES-GCM, which can be leveraged to encrypt and guarantee the integrity of sensitive data. Additional data obfuscation methods, including *data conversion* and *data filtering*, serve to reduce the sensitivity of the data or completely strip sensitive elements, respectively. Data conversion can be particularly effective for human voice recordings, converting them into voice commands upon recognition of specific voice patterns. Data filtering, on the other hand, can entirely remove unrecognised sentences. The primary advantage of data obfuscation is that it effectively restricts the distribution of sensitive data in uncontrolled environments, such as the REE and the cloud service providers.

#### 2.2.5.4 Driver Partitioning

A key aspect of TEE development is decreasing the *trusted computing base* (TCB). The latter represents the components of a system that must be trusted to enforce the system's security policies. Decreasing the TCB reduces the potential attack surface and the likelihood of vulnerabilities in the TEE. TCB reduction is usually done via *code partitioning*. This involves identifying portions of the code that manipulate sensitive data and isolating only these within the TEE; the non-sensitive part can be kept out of the TEE. OP-TEE provides some examples of how to build secure peripheral drivers but lacks a systematic approach for partitioning them into trusted and untrusted components.

This section outlines a high-level approach to assist developers in partitioning drivers for securing peripheral data in Fortress. Using a simple $\mathtt{I}^2\mathtt{S}$ peripheral driver as an example, we manually partition its code into trusted and untrusted parts. The trusted part includes the code which interacts with the peripheral's secure I/O region. For $\mathtt{I}^2\mathtt{S}$, the trusted driver handles tasks like mapping MMIO registers, read/write operations, and DMA buffer management. The untrusted part deals with non-sensitive operations like clock and power-management. Figure 2.12 provides a generic blueprint for manual partitioning in Fortress, satisfying **G4**. Advanced techniques like static data-analysis may be used for more complex code bases to pinpoint code portions interacting with sensitive regions of the peripheral.

### 2.2.6 Assessment of Fortress and Concluding Remarks

This section presented Fortress, a generic design to secure sensitive peripheral data in IoT environments by leveraging TEE technology. Fortress restricts peripheral I/O memory regions to a secure kernel-space TEE, thereby limiting access exclusively to an small segment of the peripheral driver code. This strategy effectively blocks unauthorised access from potentially compromised operating systems or hypervisors. The data is then securely transferred to a user-space TEE, where obfuscation techniques are applied prior to its transmission to an untrusted host operating system or cloud service providers. We refer to our scientific paper [200] for more details concerning our proof-of-concept implementation focusing on $\mathrm{I^2S}$, and our evaluations showing that Fortress achieves enhanced the security posture of IoT devices at a reasonable cost.

Manual partitioning can be very challenging and error-prone for large code bases, as there may be complex data paths with indirect access to peripheral data, which are difficult to identify manually. Nonetheless, in Fortress, restricting the peripheral's secure I/O region to the secure world always prevents access to peripheral data from untrusted code, even if the latter contains instructions that manipulate peripheral data. To overcome the complexities inherent in manual partitioning, techniques such as static data-analysis [49, 101] could be leveraged to properly identify all code parts that interact (directly or indirectly) with sensitive data.

We aim to extend our work along two axes: firstly by integrating static analysis techniques to streamline the code partitioning process, and subsequently by introducing machine learning classification algorithms to automate data obfuscation. The latter approach involves the use of pre-trained deep learning models within the TA to identify sensitive data in data streams, upon which appropriate obfuscation techniques may be applied.

## 2.3 Attesting MQTT Brokers within Intel SGX

Publish/subscribe (pub/sub) systems [63] have become fundamental when implementing communication of a wide range of devices, from IoT ecosystems to large-scale cloud-based services, such as the cloud-edge continuum [168]. Pub/sub systems enable efficient and scalable data distribution among distributed entities by decoupling data producers from data consumers. Given their widespread adoption, notably in cloud computing [6, 80, 129], several pub/sub systems have been proposed with the clear goal of providing additional privacy guarantees [140]. However, the nodes participating in these systems are implicitly trusted, relegating security concerns primarily to the protection of communication channels or leveraging heavyweight cryptographic primitives [96]. This limited security approach leaves data on the processing components vulnerable to potential threats, especially in decentralised and heterogeneous environments. Notably, high-privileged actors within the pub/sub nodes, *i.e.*, operating system or hypervisor, may compromise the confidentiality and integrity of data. Similarly, they could leak critical cryptographic material, *e.g.*, private keys of certificates.

Leaking certificate keys is especially concerning as these keys serve as the foundation for the authentication process among pub/sub participants, thereby putting user privacy and data integrity at stake.

In both the consumer market and cloud providers, trusted execution environments (TEEs) present a solution to strengthen the integrity and confidentiality of data in use, especially on nodes that may not be inherently trustworthy. TEEs provide *enclaves*, *e.g.*, hardware-protected memory regions, where sensitive computations are completely isolated from other software executed on the same platform. Such secure enclaves significantly elevate the security posture of systems like pub/sub, safeguarding not just the communication but also the processing and data on such nodes from malicious actors. As a keystone feature of TEEs, attestation enables a remote entity to verify the authenticity, configuration, and state of a trusted environment using cryptographic proofs, ensuring the enclave runs the intended software without being tampered with or compromised [123]. In pub/sub systems deployed over untrusted infrastructures, attestation ensures data and its processing within the TEE enclave are kept confidential and untampered.

Nonetheless, developing applications for TEEs is challenging due to their specific programming paradigms and SDKs, requiring massive efforts when writing or porting existing software [81]. In addition, while many pub/sub systems exploit TEEs to protect data in use (covered in §2.3.3), these usually tie closely to a specific TEE architecture, which is limiting in the heterogeneous environments (*e.g.*, using different CPUs) considered in this work. Beyond this, current secure communication protocols fail to transport attestation proofs, do not maintain the privacy of these messages when supported, or cannot expose X.509 certificates issued by global authorities (covered in §2.3.4). This leads to the use of ad-hoc implementations, which fall short in offering a consistent solution across the cloud-edge continuum.

We solve many major challenges associated to writing secure and portable pub/sub systems by relying on WebAssembly (Wasm), an open-standard binary instruction format. Wasm's architecture-neutral design abstracts the complexity of hardware and TEE requirements, making it particularly suitable as a compilation target for pub/sub systems in heterogeneous hardware environments, such as the cloud-edge continuum. We further protect the communication channels by extending the industry standard TLS protocol for embedding attestation evidence in the handshake while maintaining compatibility with the original specifications. This ensures the authenticity of the executing code of parties within the pub/sub system, thereby mitigating the risk of malicious entities impersonating or modifying these components. Additionally, certificate keys are safeguarded within the TEEs, preventing potential leaks.

We developed a proof-of-concept that encapsulates the full implementation of a standard pub/sub broker and a TLS library, enabling the termination of TLS channels directly in the TEE, which has been proven complex in prior work [29]. We achieve this by using Wasm as a compilation target for both software, limiting the number of code changes required to make them compile and run within TEEs. While our prototype is focused on cloud environments by leveraging Intel SGX, we outline the trusted primitives required for our proposal to be compatible with other platforms, including edge and IoT devices.

This section proposes a unified strategy that secures a standard pub/sub system using

TEEs with attestation for security, and compiled in Wasm for seamless cloud-edge communication while minimising code changes. We also propose an extension to the TLS communication protocol, facilitating the confidential exchange of attestation evidence, thereby affirming the authenticity of actors within the pub/sub system. We included these proposals in a scientific paper recently published [124], along with the detailed description of an open-source implementation of a pub/sub broker using Intel SGX for cloud environments, with a suite of benchmarks aimed at evaluating the impacts of Wasm, the TEE and attestation, in comparison with state-of-the-art work. Our evaluation reveals that our system delivers messages at a throughput that is a bit slower, yet providing portability and the robust security guarantees of TEEs.

### 2.3.1 Pub/sub Systems

A publish/subscribe system (often called pub/sub) is an asynchronous architecture for message passing that connects two different types of entities: publishers and subscribers. Publishers send messages (or events), being unaware of who is interested in receiving such messages. Messages sent are usually marked with an arbitrary type or a category. Subscribers express their interest in specific types of messages and receive them when they are published. Pub/sub is commonly used in event-driven distributed systems such as the Internet-of-things, since they simplify the communication between different entities.

Pub/sub messages are usually passed from publishers to subscribers using an intermediary broker. Brokers receive messages from publishers and forward them to subscribers who have expressed interest in the corresponding topics. Message brokers are responsible for routing and delivering messages to the appropriate subscribers. Examples of message brokers include Apache Kafka, RabbitMQ, MQTT brokers, and cloud-based pub/sub services like Amazon SNS (Simple Notification Service) or Google Cloud Pub/Sub.

One of the key advantages of pub/sub is decoupling since publishers and subscribers do not need to know each other. Pub/sub systems can also be designed to be highly scalable and fault-tolerant by using replicated, interconnected brokers. This flexible architecture allows components to be added or removed without affecting the system.

Security in pub/sub systems is dealt with when publishers and subscribers connect to their brokers. Brokers then usually implement authentication and access control before publishing or subscribing to messages. They can also establish encrypted connections (*e.g.*, TLS) to implement message privacy. By implementing attestation, we offer a stronger guarantee for all the components. Publishers, subscribers, and brokers are guaranteed to be who they say they are, and all can ensure that their counterparts execute appropriate (correct) software.

### 2.3.2 Attestation

Attestation is a security mechanism enabling an *attester* to prove its identity and integrity to an *verifier*. In the realm of Trusted execution environments (TEEs), attestation allows a TEE to prove its configuration, identity, and state to another device or service. Most TEE implementations support attestation, either built-in (as in SGX [97] and its successor TDX [158, 48], AMD SEV-SNP [7]) or by means of research prototypes [126]. The primary objective is to ensure the attester is genuine, unmodified, and trustworthy. To achieve this objective, attestation leverages proofs (*i.e.*, *evidence*), a cryptographically signed document composed of *claims*, *i.e.*, pieces of asserted information, like the hash of a code running in the TEE, *i.e.*, *measurement*. The verifier is bootstrapped with *reference values*, compared against the received claims for validation.

We focus on Intel Software Guard Extensions (SGX) [52, 137], a popular and widely used TEE architecture. In Intel SGX, attestation is paramount for establishing the authenticity and integrity of code and data inside an enclave. Intel controls a set of keys, intrinsically linked to the hardware and identity of the enclave. Such keys are inaccessible from the enclaves themselves, and are used by the processor and Intel signed software to issue trustworthy evidence. Intel is considered trusted, as they provision a subset of keys into the hardware. Within this assumption, third parties can trust evidence produced by the TEE.

#### 2.3.2.1 Communication Channel

Over the past decades, several standards have emerged to establish trustworthy communication between two entities. Among them, Transport Layer Security (TLS) stands out as the most prominent protocol for such tasks. This cryptographic protocol ensures confidentiality, integrity, and authentication for secure communication. A typical TLS session begins with a phase known as the *handshake*. This TLS handshake acts as a negotiation process, where the two parties decide on encryption settings and authenticate one another before exchanging secure data. More precisely, the latest version of the TLS handshake (1.3 at the time of this writing) is composed of *1)* the *ClientHello* sent by the client, notably containing the supporting cipher suites, key agreement and anti-replay mechanisms, *2)* the server sends the *ServerHello* response, detailing analogous parameters, while signifying the handshake's conclusion, and finally *3)* the handshake wraps up as the client reciprocates with a similar acknowledgement, accompanied by data from a higher-level protocol, that is wrapped within TLS, *e.g.*, HTTPS. Additionally, during steps 2 and 3, both entities can opt to present an X.509 certificate. These certificates authenticate entities and build trust by leveraging a chain of digital signatures from trusted Certificate Authorities (CAs). These serve as identity validators, embedding trust through a chain of digital signatures rooted in trusted Certificate Authorities (CAs). Building on this, many existing attestation protocols that bind with TLS typically augment the handshake, the certificate, or a combination of the two, facilitating the negotiation of security parameters and the exchange of attestation evidence.

#### 2.3.2.2 Channel Binding

Channel binding [28] ensures that an entity participating in a secure communication, typically via protocols such as TLS, is indeed the entity that has undergone attestation. It establishes that the communications are with the attested environment, preventing possible relay attacks where a malicious party might relay attestation challenges to a genuine system and then claim the legitimate evidence as their own. To secure trusted communication, attestation evidence is typically integrated into the early phases of the communication protocol. However, combining attestation with these protocols introduces challenges, including increased latency and additional trade-offs like binding the system to a particular TEE technology.

The solution we propose refines the TLS protocol with minimal enhancements. Unlike traditional approaches that embed attestation in custom fields in the broker's X.509

| | [150] SCBR | [25] PubSub-SGX | [189] MagikCube | [161] MQT-TZ | [147] Pei *et al.* | This work |
|---|---|---|---|---|---|---|
| Comm. protocol | TLS | TLS | TLS | TLS | ? | TLS |
| Fully enclaved broker | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ |
| Peer authentication | ✗ | ✓ | ✓ | ✓ | ? | ✓ |
| Peer attestation | ✗ | ✗ | ✗ | ✗ | ? | ✓ |
| Broker authentication | ✓ | ✓ | ✓ | ✓ | ? | ✓ |
| Broker attestation | ● | ✓ | ✓ | ✗ | ? | ✓ |
| Persistence of messages | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| Idiomatic pub/sub arch. | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Open source | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ |
| TEE technology | SGX | SGX | SGX | TZ | SGX | *Agnostic* |

TEEs references: SGX (Intel SGX), SNP (AMD SEV-SNP), TDX (Intel TDX), TZ (Arm TrustZone)

✓, ●, ✗ mean fully, partially and not implemented, respectively. ? denotes not disclosed details. Features description: *1) Communication protocol*: protocol used by peers to interact with the broker. *2) Fully enclaved broker*: broker operates within the TEE instead of only securing specific components within the secure environment. *3) Peer authentication*: broker authenticates the peers while establishing communication. *4) Peer attestation*: broker attests the peers while establishing communication. *5) Broker authentication*: peers authenticate the broker while establishing communication. *6) Broker attestation*: peers attest the broker while establishing communication; ● means that only publishers are attested. *7) Persistence of messages*: system's capability to store messages for future delivery (*e.g.*, when subscribers might be temporarily offline). *8) Idiomatic pub/sub architecture*: system adheres to the principles of the pub/sub paradigm. *9) Open source*: implemented solution is freely available to the public via an open-source repository. *10) TEE technology*: denotes the TEE used by the proposed system, or its capacity to operate agnostically across various TEEs.

*Table 2.1. Comparison of the state-of-the-art pub/sub systems shielded by TEEs.*

certificates, our method leverages a custom TLS 1.3 encrypted extension. This approach reduces the need for additional communication roundtrips between the client and server. It preserves the broker's ability to use certificates issued by recognised CAs, an aspect overlooked in previous studies.

### 2.3.3 Pub/sub with TEEs

In the dynamic world of distributed systems, pub/sub mechanisms have consistently gained traction, acting as the core foundation for a variety of applications and architectures. Many approaches have been conducted in research to bring dependability and safety regarding pub/sub systems, and in many different directions [187]. An explored aspect is encryption schemes of data transferred through the brokers, preserving the privacy of the communicated information [93, 136, 35, 41, 118, 70, 36]. Cryptographic-based privacy protection schemes focus on encrypting events and subscriptions, and then performing ciphertext matching between them. However, they often suffer from scalability issues as matching time complexity grows with the number of subscriptions, leading to diminishing performance.

More recently, researchers have investigated the potential of Intel SGX as a secure environment for confidential data processing. Leveraging TEEs has shown potential for enhanced performance over cryptography-based methods, as highlighted by SCBR [150]. Their study details a custom content-based routing engine operating within an SGX enclave. PubSub-SGX [25] introduced a scalable approach, using a load balancer that manages multiple matchers, each operating within individual enclaves. Following this, MagikCube [189] added an authentication service to the pub/sub system, thereby enhancing broker trust among publishers and subscribers using SGX. Finally, Pei *et al.* [147] further refined this paradigm by optimising subscription matching times using cryptographic methods, with SGX facilitating comparison tasks but does not disclose how secrets are exchanged. In contrast with previous work, our proposal prioritises establishing an initial trust across all pub/sub participants by mutual attestation. We do it by encapsulating conventional pub/sub systems within TEEs, ensuring genuine execution environments and trustworthy implementations.

Other prior studies have chosen a different strategy using TrustZone, Arm's TEE. In this setup, the device is divided into the *normal world* (the standard OS) and the *trusted world* (the TEE). MQT-TZ [161] migrated the broker's data management component within this trusted world. Publishers and subscribers negotiate symmetric keys with the broker, which are generated inside the TEE. This approach ensures that data is encrypted during transmission. Conversely, our proposal enhances the threat model of MQT-TZ by relying on entity attestation in the pub/sub system and hosting the TLS endpoint directly within the TEE, avoiding handling cryptographic materials outside the TLS protocol. Table 2.1 offers a comprehensive summary of state-of-the-art proposals for securing pub/sub systems with TEEs, focusing on features relevant to our study.

### 2.3.4 Communication Protocols using Attestation

Research has extensively explored the integration of secure communication protocols with attestation. While our focus primarily lies on solutions leveraging TEEs, we

| | [91] SGX EPID | [162] Shepherd *et al.* | [126] WaTZ | [102] RA-TLS | [82] Palæmon | [138] TSL | This work |
|---|---|---|---|---|---|---|---|
| Baseline protocol | Custom | Custom | Custom | TLS | TLS | TLS | TLS |
| No change in TLS spec. | — | — | — | ✓ | ✓ | ✓ | ✓ |
| Attestation privacy | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ |
| Mutual attestation | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ |
| Evidence per session | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| Endpoint in enclave | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Attestation privacy | ✗ | ● | ✗ | ● | ? | ● | ✓ |
| TEE-agnostic | ✗ | ● | ● | ✗ | ✗ | ● | ✓ |
| Support global CAs | — | — | — | ✗ | ✗ | ✗ | ✓ |
| Open-source | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ |
| TEE technology | SGX | TZ | TZ | SGX | SGX | *Agnostic* | *Agnostic* |

TEEs references: SGX (Intel SGX), SNP (AMD SEV-SNP), TDX (Intel TDX), TZ (Arm TrustZone)

✓, ✗ mean fully and not implemented, respectively. ? denotes not disclosed details. "—" denotes not applicable comparisons. Features description: *1) Baseline protocol*: protocol upon which the proposal is built. *2) No change in TLS specification*: proposal respects the TLS specifications, ensuring compatibility with pre-existing TLS implementations. *3) Attestation privacy*: protocol maintains confidentiality of attestation evidence. *4) Mutual attestation*: communicating entities engage in a mutual attestation process. *5) Evidence per session*: each communication session is uniquely associated with specific attestation evidence. *6) Attestation Privacy*: all attestation-related messages are encrypted; ● means that only a portion of the messages remains confidential. *7) TEE-agnostic*: independent of any specific programming language or TEE-specific SDK; ● means theoretical approach proposed, but no agnostic implementation. *8) Endpoint in enclave*: endpoint application fully resides within the TEE. *9) Support global CAs*: broker-displayed certificates can be vouched for by globally recognised CAs. *10) Open source*: implemented solution is freely available to the public via an open-source repository. *11) TEE technology*: denotes the TEE that the attestation API currently supports, or its capacity to operate agnostically across various TEEs.

*Table 2.2. Comparison of the state-of-the-art channel binding solutions.*

also recognise the significant contributions from prior work that leveraged TPMs as trust anchors [75, 167, 72, 166, 24, 198, 31, 146, 185, 184]. Readers can refer to [102, 138] for a comprehensive review of these works. Our analysis omits explicitly works that discuss attestation through TEEs but do not bind attestation evidence to a communication channel [120, 53, 68, 191].

Intel proposed a remote attestation protocol for key exchange based on SIGMA [103, 91], binding SGX enclaves with communication channels. Subsequent solutions aimed to establish trusted communication channels with enclaves using custom message exchanges [162, 108, 126]. While these initial efforts in remote attestation provided valuable insights, their custom nature makes them challenging to integrate into existing software. In contrast, our approach harnesses TLS, the leading industry standard for secure communication.

More recently, further research [178, 82, 138, 102, 139] have suggested using TLS for communication and modifying the X.509 certificates in order to include additional fields related to attestation. Although this method takes advantage of the protocol's standardisation to address the earlier concern, it ties the attestation mechanism directly to the certificates exposed by the endpoints. This direct connection implies that certificates must be dynamically generated, which restricts them from being signed by global CAs like *Let's Encrypt*, commonly used for domain validation certificates. Opting for a different route, we used the encrypted extensions of the TLS protocol for carrying evidence of the server. Consequently, our approach is compatible with certificates endorsed by global CAs. This is particularly beneficial if the endpoint owns a separate network-level identity, like a DNS or domain name. Table 2.2 offers a comprehensive summary of cutting-edge research dedicated to binding communication channels with attestation, focusing on features relevant to our study.

### 2.3.5 Threat Model

Our approach relies on a few key trusted components, which are essential for our system to be deemed trustworthy. We further discuss these elements in the remainder of this section.

**TEEs** Our proposal leverages the protection offered by TEEs for securing the execution of applications and enforcing strong isolation against powerful attackers, such as the OS or the hypervisor. Given that our proposal is TEE-agnostic, we highlight the minimal requirements to uphold trust in the pub/sub system and attestation mechanism. We assume the application code can be inspected but cannot be subverted. Data in use remains confidential and cannot be read unless granted by the TEE. The hardware and software strictly required to run a TEE instance are considered trusted. Furthermore, the TEE offers a remote attestation mechanism backed by genuine entities responsible for validating the trustworthiness of attestation evidence. Although we do not address side-channel or denial-of-service attacks [43, 192, 67], there exist measures for these in various TEE designs [3].

**OS** The OS follows an *honest-but-curious* threat model, posing no threat to the trusted environment but interested in gathering sensitive information. Consequently, it can monitor all communication within the pub/sub system. Assuming the OS starts to behave maliciously, the trusted computing base (TCB) remains confidential and doesn't malfunction, though it might become unresponsive. The applications running in the TEE are carefully developed to ignore abnormal responses and abandon execution in such cases.

**Wasm** We presume that the Wasm runtime is implemented correctly and does not contain vulnerabilities. The Wasm runtime acts as a shim library by encapsulating Wasm applications and uses trusted APIs from the TEE SDK for system interactions or sanitises the interaction with the untrusted OS when no secure option is available. While side-channel attacks might target Wasm and TEEs [152], they fall beyond the scope of this study.

**Cryptography** As we embed a TLS library and enhance it to integrate attestation concerns, we suppose the correct implementation of the cryptographic ciphers and operations. Additionally, we also presume that standard cryptographic techniques cannot be subverted and are hardened against side-channel attacks.

### 2.3.6 Security Requirements and Trusted Primitives

We propose a series of requirements for establishing trusted communication channels between the actors of pub/sub systems:

**(SR1): Support global CAs certificates**: brokers shall support exposing certificates issued by globally recognised CAs.

**(SR2): Trust assurance**: pub/sub actors shall attest the TCB of the participant they connect with and must be similarly verified in return.

**(SR3): Channel and attestation bindings**: communication channels must be linked to newly created attestation evidence, preventing replay or collusion attacks [138].

**(SR4): Attestation privacy**: attestation information shall remain confidential between the endpoints of a given communication channel.

**(SR5): Pub/sub privacy**: all the data bound to the pub/sub system shall be inaccessible to outside actors, including the OS, kernel and hypervisor of the broker and peers.

**(SR6): Pub/sub narrow scope**: the peers shall only publish or subscribe to the topics as required for their needs.

We identified a core set of trusted primitives that any system must support to host our proposal securely. These primitives are provided by the secure environment or the Wasm runtime:

- **Isolated execution context**: this ensures the runtime remains secure and unaffected by any other applications running concurrently on the system. All major TEE manufacturers support at least one Wasm runtime, though the isolation paradigm and threat model vary.

- **Attestation capabilities**: the TEE must expose two primitives to the Wasm runtimes: *generate* and *verify*. The former primitive generates evidence for proving the trustworthiness of the secure environment with additional data attached, such as a nonce and a public key, while the latter confirms the validity of this proof.

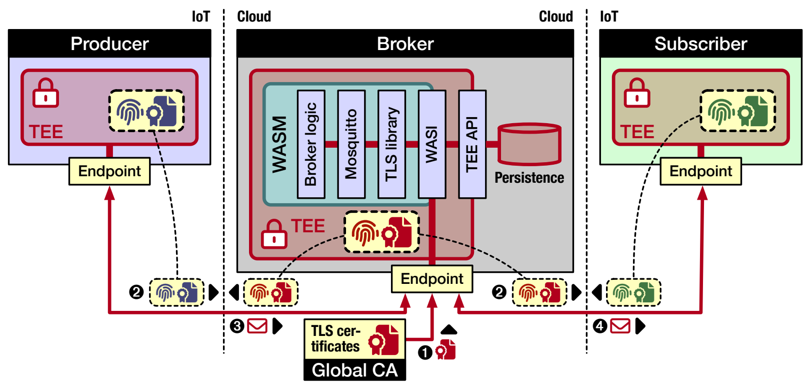


*Figure 2.13. The overall architecture of our proposal. (📜, 🫆) mean X.509 certificate and attestation evidence, respectively. The colours of these icons correspond to the actor owning them.*

- **Network communication**: the Wasm runtimes require access to a network API that handles socket operations and the transfer of data.
- **File system access**: if the pub/sub broker needs to store publications for future delivery, the system should offer secure ways to save and access these files.

Other concerns, such as encryption and pub/sub protocol logic, are platform-independent and addressed using dedicated software compiled into Wasm.

### 2.3.7 Architecture Overview

We designed our proposal as a versatile system capable of running on numerous processor architectures. The system isolates security-sensitive pub/sub operations of peers and brokers, ensuring the authenticity of connecting machines using TEEs and mutual attestation. A key aspect of our design is the adoption of Wasm, facilitating cross-platform support across various TEE architectures. More specifically, we rely on trusted Wasm runtimes [125, 126], to host a secure pub/sub system with its associated dependencies such as TLS libraries, thus covering the large spectrum of the cloud-edge continuum.

Figure 2.13 illustrates the key components and entities within our pub/sub solution. Serving as the central hub, the broker first acquires a certificate from a global CA for its TLS endpoint (❶). Peers then initiate secure communication with the broker via our enhanced TLS handshake, performing mutual remote attestation by exchanging their X.509 certificate for authentication and attestation evidence (detailed in Section 2.3.8). This ensures that the broker and peers are trustworthy (❷). The publisher generates data and transmits it to the broker's TEE (❸). The broker, in turn, relays this data to the subscriber's TEE (❹). Both the publisher and the subscriber use the same technology stack in their respective TEEs, although this detail is omitted in the figure for clarity.

### 2.3.8 Attesting Communication Channels

We enhanced the TLS protocol to integrate the exchange of attestation evidence when a peer is communicating with the broker. This exchange of information occurs in the TLS handshake, as depicted in Figure 2.14. Our enhancements are highlighted in bold.

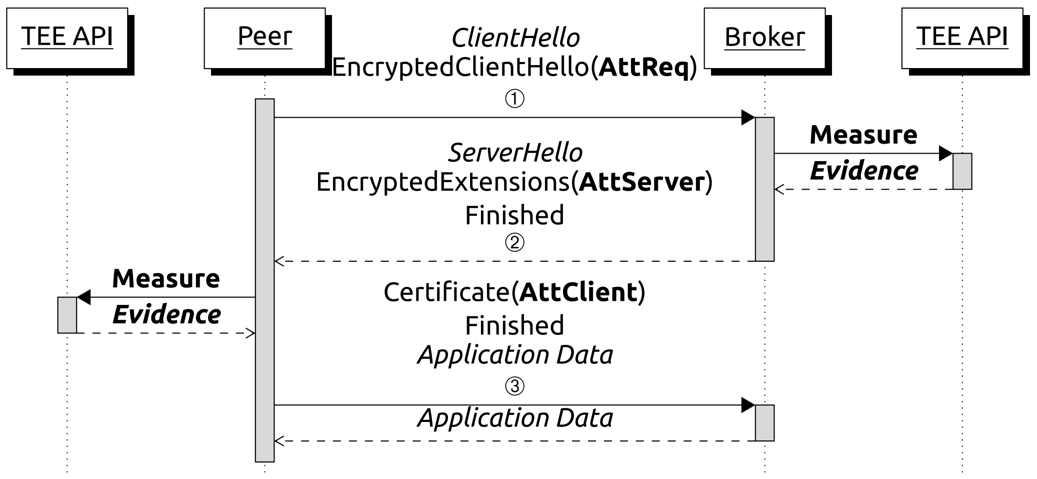


*Figure 2.14. Enhanced TLS 1.3 handshake with attestation. New elements are mentioned in bold.*

**Attestation protocol** The first message of the handshake (①) is sent by the peer, which is comprised of an encrypted attestation request (**AttReq**), indicating that the peer is establishing a TLS channel requiring attestation. Contrary to prior work, our protocol does not specify TEE architectures in this message, as our solution supports the verification of all the types of TEEs that can be used in the broker. Similarly, our protocol is the only proposal that studies the encryption of the attestation request, as further explicated below.

When received, the broker answers with a message (②) composed of its evidence (**AttServer**), freshly generated and bound to the TLS session. It is worth noting that we deliberately chose not to embed attestation data within the broker's X.509 certificate so the endpoint can use certificates endorsed by globally recognised CAs, satisfying **(SR1)**.

Subsequently, the peer verifies whether the evidence of the broker is genuine and checks if the evidence is bound to the current TLS session (to counter reuse attacks). Moreover, it compares the code measurement of the broker to a known reference value, indicating that the code of the broker is trusted. Since a global CA issues the broker's X.509 certificate, the peer verifies that the DNS or domain name of the broker matches the identifier exposed in the X.509 certificate [157]. If these conditions are fulfilled, the peer issues its evidence (**AttClient**) and sends it to the broker (③), also bound to the TLS session and embedded within the peer's X.509 certificate. We had to rely on a custom X.509 extension for the peer, as TLS 1.3 does not provide an extension point for the last message of the handshake.

Lastly, the broker also verifies whether the evidence of the peer is genuine, and checks if the evidence is bound to the current TLS session. Similarly, the broker ensures that the code measurement of the peer matches a known reference value, indicating that the code of the peer is trusted. Once completed, the peer and the broker mutually attest themselves, ensuring their respective TEE is trustworthy, satisfying **(SR2)**. As a result, the TLS handshake is ended, and the applications may start pub/sub communication.

**Binding the handshake with attestation evidence** Integrating the attestation mechanism in the TLS handshake has many benefits: the execution time is optimised since no additional round-trips are necessary, and the TLS 1.3 standard is respected by using extension points as designed by the protocol. Besides, we strongly bind freshly generated evidence to individual TLS sessions, using unique TLS keying materials computable by both parties (RFC 5705) [155], satisfying **(SR3)**. This prevents replay and collusion attacks, which would affect published past evidence otherwise. Since our pub/sub system and TLS library are compiled in Wasm, evidence binding within TLS handshakes is portable across different TEE architectures.

**Securing the attestation information** We use three distinct encryption mechanisms to ensure the confidentiality of the attestation information, to comply with **(SR4)**. First, we opted to use the recently-introduced *encrypted client hello* (ECH) for TLS [156], which is currently an IETF draft to communicate early information while preserving its privacy in the *ClientHello* message. To the best of our knowledge, we are the first to leverage this draft to protect the attestation request (**AttReq**) in the TLS handshake. In a nutshell, the *ClientHello* message is split into two parts: the outer message, which is in plain text, and the inner message, which is encrypted using a public key, typically distributed using DNS infrastructure. Second, we rely on the TLS 1.3 encrypted extension in the broker reply (*i.e.*, the `ServerHello` message), so the broker's attestation evidence remains confidential. This has been made possible because the two parties can derive a shared secret for encryption up to this point. Finally, the peer's certificate, which contains the peer's evidence, is also protected, since the entire third message of the handhake is encrypted by design.

### 2.3.9 Securing pub/sub Systems

We leverage TEEs and trusted Wasm runtimes for implementing our pub/sub design, ensuring strong hardware isolation of both code and data. This design remains versatile, working with various TEE architectures as long as they offer the trusted primitives for the Wasm runtimes (as in §2.3.6). When paired with the mutually attested TLS protocol, we shield data in use and in transit, satisfying **(SR5)**. As a pub/sub software, we selected Mosquitto [110], a well-known and open-source message broker that implements the latest version of the MQTT protocol. We have chosen Mosquitto because it is lightweight and suitable for use on all devices, from low-power IoT devices to cloud servers. Besides, we used WolfSSL, an embeddable cryptographic library, enabling the host of the TLS termination directly in the TEE, so external systems cannot eavesdrop on the communication, nor alter the code that maintains the endpoint. We needed to compile Mosquitto in Wasm, which required slight modifications to the source code. Regarding WolfSSL, we reused one of the extension works of a trusted runtime [125], which already compiled this library in Wasm.

**System interactions** Mosquitto, like most software, is required to perform system calls for interactions with the outside world. Typically, this is the case for the socket API when exposing the TLS endpoint. For such usages, we rely on WASI, which translates the system calls to the underlying OS seamlessly. As the private and session keys of TLS are located within the TEE, the communication remains confidential against eavesdropping attempts, even when system calls are monitored. Besides, Mosquitto can persist undelivered messages in a database for later transmission to offline subscribers. The WASI specification includes a file system API for data storage, while TEEs typically provide means to save files via a trusted API securely. In contrast to the socket API, Wasm runtimes use that TEE API to transparently encrypt files, ensuring the confidentiality of the stored messages.

**Securing and narrowing the pub/sub data access** Our proposal ensures that brokers and peers are trustworthy as they are mutually attested. As such, we inherently restrict adversaries to propagate and observe messages going through the pub/sub system. Nonetheless, we propose to reduce further the scope of peers using access control lists (ACLs), which is a built-in functionality of Mosquitto. When paired with the X.509 certificates presented in the TLS handshake, this feature enables to authenticate peers without requiring additional usernames or passwords, as this is typically the case using Mosquitto. This narrowed publication and subscription scope satisfies **(SR6)**. Future work may further adapt the ACLs to be bound to evidence, provided that code measurements differ from each actor.

**Use cases** Our approach is applicable to a variety of distributed pub/sub systems, which supports scalable communication by decoupling publishers from subscribers and ensures secure messaging through mutual attestation. In VEDLIoT, this is demonstrated in collaboration with Siemens for secure data processing at edge nodes and capturing results on an MQTT messaging bus, also using the Byzantine Fault-Tolerant (BFT) attestation provided by SIRE (see Chapter 5 for ensuring trust in the attestation reference values [182]. To balance the need for compactness with computational capability, we used Intel NUCs as edge nodes, which are small, efficient PCs equipped with processors that support Intel SGX. This allows the publisher on the edge nodes to verify whether the broker is genuine based on evidence and the code hash contained in that payload. This code hash is then compared to the trusted reference values stored in the BFT-resilient system.

### 2.3.10 Concluding Remarks about Attested pub/sub

Recent evolution in TEEs by leading CPU manufacturers highlights the growing trend in executing software within untrusted environments while processing increasingly sensitive data. The pub/sub model stands out as an effective mechanism to scale and distribute computations across varied architectures, and Wasm emerges as a suitable common environment for such tasks. However, a gap remains in establishing standardised protocols and tools that leverage the rapid research breakthroughs in trusted execution within the cloud-edge continuum.

Addressing this, we proposed a secure and attested pub/sub system compatible with most of the state-of-the-art TEEs. We achieve this by using Wasm and trusted runtimes running in these TEEs. To incorporate mutual attestation in network communications, we suggest enhancing the TLS protocol. This modification embeds attestation evidence through extension points in the TLS handshake, preserving the original standard.

Our evaluation can be found in our scientific paper [124] and demonstrates that the security and portability improvements introduced by our approach effectively balance the additional overheads, mostly related to the TEEs. Moreover, leveraging Mosquitto's capability to distribute brokers among peers can further optimise the system, leading to a scalable and secure architecture suitable for large and real-world applications. Our implementation is freely distributed as an open-source project.[2]

[2] https://github.com/JamesMenetrey/unine-opodis2023

# 3 Trusted Certification of IoT Devices

This chapter is the continuation of the work already presented in the previous Deliverable D5.3, Chapter 4. To keep this deliverable self-contained, we will repeat some of the content, but notable additions to the previous work include details on how to realise our proposed certification solution with X509-based certificates, a comparison of the network overhead of the different versions, and a formal analysis of some of the security properties claimed by our protocol with the Tamarin Prover.

## 3.1 TruCerT: Definitions and Concepts

This section provides an overview of the stakeholders involved in the TruCerT certification process and TPM concepts. We envisage a certification process inspired by EU Cybersecurity Certification Framework (shown in Figure 3.1); therefore, same terminologies and names of entities are used to allow easy mapping of TruCerT with the upcoming Framework. However, it is also pertinent to state that these entities/stakeholders are common for any regional or national certification scheme.

**IoT Device Manufacturer**

*IoT device manufacturers* and vendors are entities responsible for implementing the security features, certification capabilities, providing technical and non-technical support to end-users throughout the device lifecycle.

**Industry Group**

An *industry group* is usually a membership-based, non-profit organization of partners who collaboratively identify challenges of the industry and potential solutions in a particular domain. They work towards vendor-neutral initiatives and are an important source of shared knowledge, best practices and advice.

**End-users**

The *end-users* of devices are either direct consumers or solution providers responsible for deployment of the IoT devices and networks.

**CAB**

A *Conformity Assessment Body (CAB)* is responsible for performing the assessment of IoT devices for certification or re-certification. A CAB is usually an independent third-party which is neither the manufacturer nor the provider of the IoT device under assessment.

**NAB**

In order to include third-party CABs in the chain-of-trust, they need to be accredited. A *National Accreditation body (NAB)* is responsible for accrediting these CABs On fulfillment of a set of requirements. An accreditation lasts for 5 years or any pre-set period of validity.

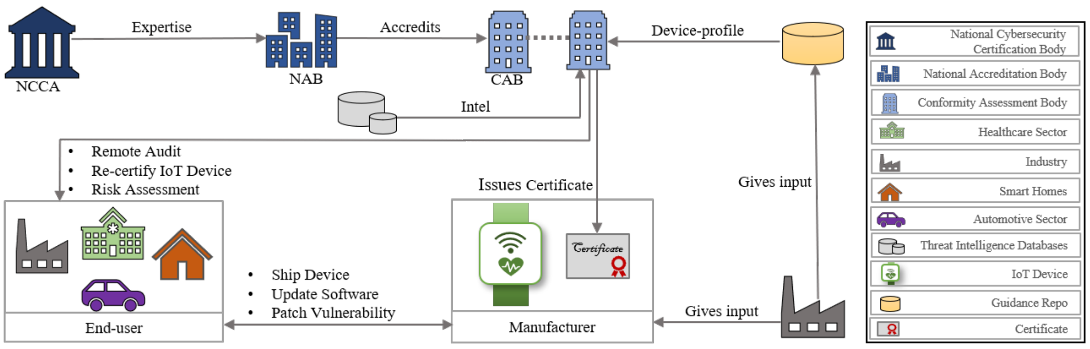


*Figure 3.1. The entities and stakeholders of TruCerT process adopted from the EU Cybersecurity Certification Framework. The IoT device manufacturers are responsible for implementing certification capabilities in devices and initiate certification. The CAB performs the audit and certification protocol (i.e. TruCerT) with the IoT Device based on the Device-profile received from guidance repo. It is also responsible for risk assessment and re-certification after IoT device's deployment. The industry group is an important source of shared knowledge, best practices and advice. The NAB is charged with responsibility of accrediting the CAB/s and the NCCA oversees the entire process and provides expertise to the NAB.*

### NCCA

A *National Cybersecurity Certification Authority (NCCA)* is a critically important entity stated in the EU Cybersecurity Act. It is held responsible for overseeing various aspects of the entire certification process. It provides the NABs with expertise and information and validates CABs on the requirements to authorize them to perform assessment and certification.

Trusted Computing is a technology that provides guarantees to the user about what software is running on device and whether the behaviour of the software follows the intended patterns. Trusted Platform Module (TPM) [83] is one such standardised technology, whose primary scope is to assure platform integrity by ensuring the behaviour of software on target device. This is done by *measuring* the software and hardware on device. Each TPM is equipped with a unique key pair (e.g. RSA keys) where the private key never leaves the TPM chip and is never exposed to any component on the device. Devices equipped with TPM, hence have the ability to establish root-of-trust starting from the TPM-unique key. Each software stage during the boot-up process performs measurements of the software components in the next phase. These *measurements* are digests of the memory regions and are extended into Platform Configuration Registers (PCR) whose size depends on the selected algorithm. The TPM merges the measurement with the measurement of the previous phase. This process is known as *Measured Boot*.

## 3.2 TruCerT Protocol

The security profile of an IoT device is defined by its hardware and software components. The scope of this paper is the security certification of the software and does not encompass the security of individual software components. TruCerT only focuses on audit of the device's software-state (i.e. software stack) and perform trusted certification. We assume a composite certification scheme [64] which would allow TruCerT to issue a certificate based on the audited software-state. Finally, the aim of TruCerT is to assign `substantial` or `high` assurance level certificates as required

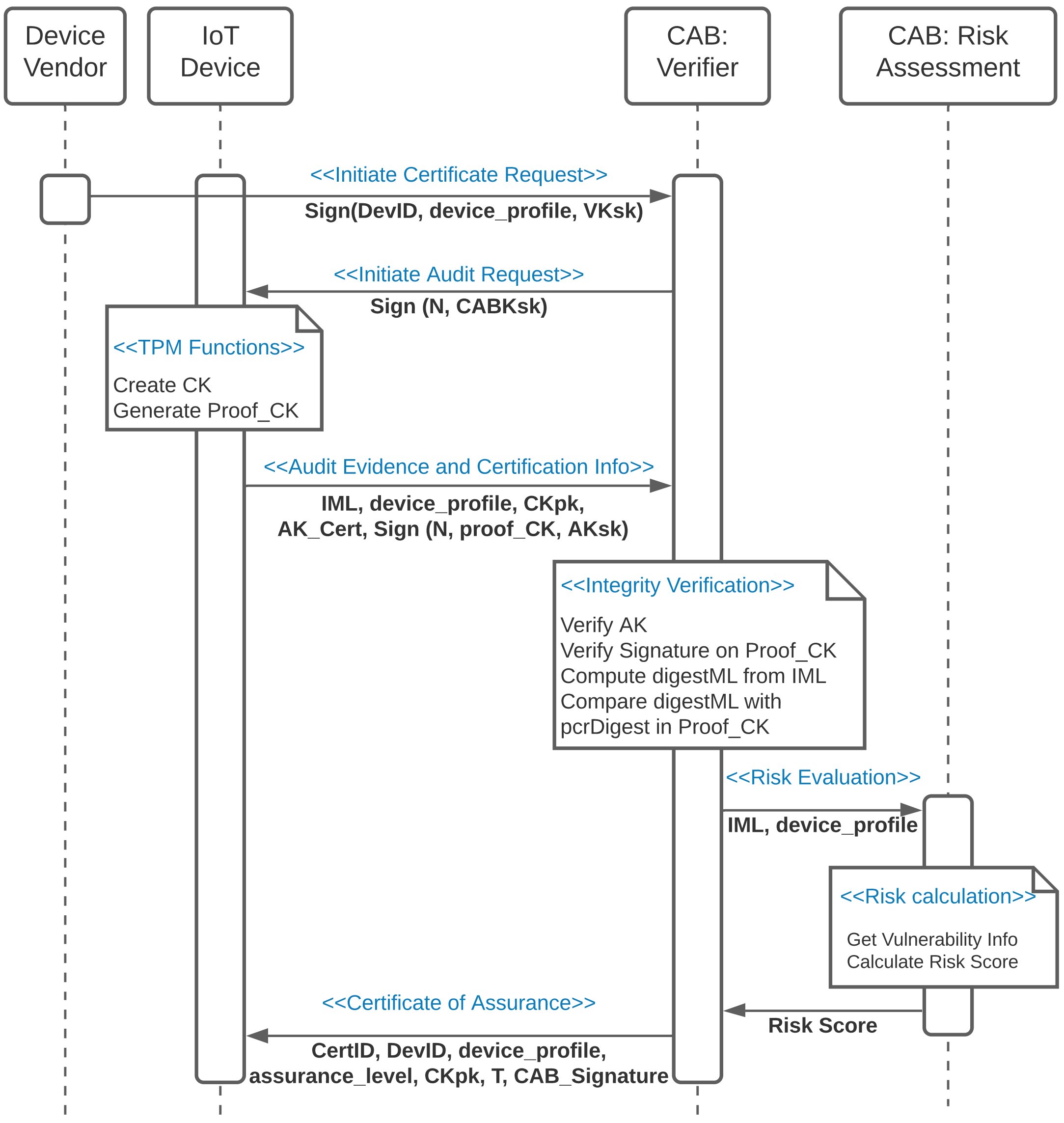

Device Vendor
IoT Device
CAB: Verifier
CAB: Risk Assessment
<<Initiate Certificate Request>>
Sign(DevID, device_profile, VKsk)
<<Initiate Audit Request>>
Sign (N, CABKsk)
<<TPM Functions>>
Create CK
Generate Proof_CK
<<Audit Evidence and Certification Info>>
IML, device_profile, CKpk, AK_Cert, Sign (N, proof_CK, AKsk)
<<Integrity Verification>>
Verify AK
Verify Signature on Proof_CK
Compute digestML from IML
Compare digestML with pcrDigest in Proof_CK
<<Risk Evaluation>>
IML, device_profile
<<Risk calculation>>
Get Vulnerability Info
Calculate Risk Score
Risk Score
<<Certificate of Assurance>>
CertID, DevID, device_profile, assurance_level, CKpk, T, CAB_Signature

*Figure 3.2. The `Device Vendor` initiates TruCerT and the `CAB` then runs the certification process with the `IoT Device`. After `IoT Device` attestation and risk evaluation, the `CAB` generates a `Certificate of Assurance`.*

by the EU Cybersecurity Act. On the technology side, TruCerT is based on TPM 2.0. From a broad class of IoT devices, we only consider IoT devices supporting TPM 2.0, e.g., Industrial IoT devices [130], and IoT Edge devices [65, 66]. The upcoming sections describe the attributes of the certificate, the process of certification and certificate usage. It is important to clarify that the scope of this work is strictly confined to the certification process. Vulnerability assessment and risk evaluation are generally considered in TruCerT but not focused. TruCerT comprises of the following steps (Fig. 3.2):

### 3.2.1 Initiate Certificate Request

The `CAB` provisions remote audit and certification service through TruCerT which is used by the `Device Vendor` to initiate certification for individual IoT devices. TruCerT is only available to IoT devices which are TPM 2.0 compliant; i.e, they implement measured boot. The `Device Vendor` initiates a request for certificate by providing the Universal Resource Identifier (URI) of the `IoT Device` (i.e. `URIdev`).

### 3.2.2 Initiate Audit Request

The CAB runs the TruCerT protocol directly with the IoT Device for RIV. The Verifier module of the CAB generates a unique nonce (N) and requests the IoT Device for its attestation data.

### 3.2.3 Audit Evidence and Certification Info

The certification protocol relies on a trustworthy way of reporting the current state of IoT Device software-state to the CAB. TruCerT uses TPM sealing capabilities to bind the issued Certificate of Assurance ($Assurance\_Cert$) with the audited state of the IoT Device, i.e. the certificate is automatically invalidated if IoT Device software-state is changed. To achieve this, a TPM-based asymmetric Certificate-Binding Key ($CK$) is generated. This key is an important element of $Assurance\_Cert$. We use Elliptic Curve Cryptography (ECC) based asymmetric keys which are well supported by TPM 2.0 due to its algorithm agility [45]. Creating this key pair using TPM2_Create() command and passing a list of PCR(s) through the TPML_PCR_SELECTION structure parameter (Listing 3.1) allows the generated key value to be dependent on selected PCR values.

```
//Declare in and out parameters for CK
TPML_PCR_SELECTION creation_pcr;
TPM2B_PUBLIC *outPublic;
TPM2B_PRIVATE *outPrivate;
TPM2B_CREATION_DATA *creationData;
TPM2B_DIGEST *creationHash;
TPMT_TK_CREATION *creationTicket;

//Generate CK keypair
tpm2_create(&creation_pcr, //in parameters
      &outPrivate, &outPublic, //out parameters
      &creationData, &creationHash, //out parameters
      &creationTicket); //out parameters
```

*Listing 3.1. TPM 2.0 command to generate the Certificate-Binding Key*

As the selected PCR(s) contain a digest value that is computed and stored during measured boot and represent the software-state of the IoT Device, using these PCR values in $CK$ creation enforces TPM to validate this software-state whenever $CK$ is used. The successful creation of $CK$ returns the keypair outPublic ($CK_{pk}$), outPrivate ($CK_{sk}$), creationData together with its hash creationHash. The creationData contains a list of selected PCR(s) (pcrSelect) and their digest (pcrDigest). Moreover, a creationTicket is also returned. The $CK_{pk}$ is included in the audit evidence whereas its corresponding $CK_{sk}$ resides in the TPM. The creationData together with the creationHash comprise the *proof_CK* and are also included in the audit evidence.

The IoT Device forwards its TPM-generated audit evidence whose fields are listed below:

- $device\_profile$: This identifies the usage context of the IoT Device (e.g. home consumer, industry 4.0, healthcare) and is to be included in the $Assurance\_Cert$. This information is essential for the End-user viewing the $Assurance\_Cert$ and helps in applying appropriate authorization policies for IoT devices.
- $IML_{dev}$: The Integrity Measurement Log ($IML_{dev}$) ia a list of all software modules and their hashes, loaded in the IoT Device. $IML_{dev}$ will be used to evaluate the

health/integrity of the `IoT Device` software-state.

- $AK\_Cert$: This is the Attestation Key Certificate issued by an Attestation CA (trusted third-party) which certifies that the $AK$ is a valid TPM key generated by an authentic `IoT Device`, and is used for attestation of TPM-generated data (e.g. PCR, TPM keys, etc.). It contains the public counter-part of TPM's unique key $AK_{pk}$ to be used by the `CAB` to validate TPM-signed data. The corresponding $AK_{sk}$ is used by the `IoT Device` to sign data, this key is used strictly for signing TPM-generated values. The identity information of the `IoT Device` ($Dev_{ID}$) is also included in the $AK\_Cert$.

- $CK_{pk}$: This key, as discussed earlier, is locked to the `IoT Device`'s software-state reported in the $IML_{dev}$.

- $proof\_CK$: TPM-generated proof of the $CK$. This information is signed by TPM's unique key $AK_{sk}$. The $proof\_CK$ contains the proof of PCR(s) used in the binding of $CK$ to the software-state. The `pcrDigest` in $proof\_CK$ is compared by the `CAB` against the self-computed `pcrDigest` from the received $IML_{dev}$.

- $N$: If the verifier's Nonce *N* does not match the device's *N*, it indicates a replay attack.

On receiving the audit evidence, the `CAB` performs multiple verification steps before trusting the `IoT Device`. The `CAB` evaluates the integrity of the `IoT Device` by checking the received $IML_{dev}$ and performing the following checks:

1. The $AK_{pk}$ is received in $AK\_Cert$ which is validated by verifying the signature of Attestation CA.

2. The `CAB` needs proof that $CK$ is a TPM-resident key which reflects the `IoT Device`'s current software-state. This is obtained by using the `TPM2_CertifyCreation()` command (Listing 3.2) which only certifies TPM-created objects (i.e. $CK$ in our case). The command takes the $CK$ handle (to identify it), its `creationHash` and `creationTicket` and compares them against the self-computed `test ticket` for $CK$. After checking validity of the ticket, TPM creates `proof_CK` structure which contains the `creationData` and `creationHash` of the $CK$ and is signed by $AK_{sk}$. This proves that $CK$ is a TPM-resident key [177, 176, 175].

3. The correctness of $CK_{pk}$ is validated by checking the signed `proof_CK`. This is done by comparing the $CK_{pk}$ digest in the `proof_CK` structure with the value computed from the received $CK_{pk}$. Since, `proof_CK` is a TPM-generated attestation information, therefore `CAB` can trust that $CK_{pk}$ is the correct key.

4. The `proof_CK` structure also validates that $CK_{pk}$ was created with certain TPM PCR(s) to bind the $CK$ to the software-state. The PCR information is obtained from the `creationData` included in `proof_CK`.

As a result of the verifications, the `CAB` can establish trust that (a) the request is sent by a legitimate `IoT Device`; and (b) the received $CK$ is a TPM key which reflects the software-state of `IoT Device` represented by $IML_{dev}$.

```
//Declare out parameters for proof_CK
TPM2B_ATTEST *proof_CK;
TPMT_SIGNATURE *signature;

//Generate CK's proof i.e. proof_CK
tpm2_certifyCreation(&CKhandle, &creationData,
          &creationHash,&creationTicket  //in parameters
          &proof_CK, &signature); //out parameters
```

*Listing 3.2. TPM 2.0 command to generate proof of $CK$ being TPM-resident*

### 3.2.4 Risk Evaluation

After successful RIV, the audit is complete and the CAB analyzes the software-state for risk evaluation by comparing it to a reference list $RML_{dev}$ provided by the manufacturer. The $RML_{dev}$ contains a list of software that the `IoT Device` was shipped with and the respective hashes. Another common approach for calculating the device's health is by comparing the software list to sources like MalwareTextDB [1] which lists millions of software and calculates the status (GOOD, MALWARE, UNKNOWN, SUSPICIOUS). Such an analysis can be used to assign an assurance level (`Basic, Substantial, High`) to the $device\_profile$ as is proposed by [27, 26]. The $Assurance\_Cert$ is generated after this evaluation and includes the calculated assurance level.

### 3.2.5 Certificate of Assurance

The $Assurance\_Cert$ is generated after verification of the audit evidence and risk evaluation of the `device_profile`. Critical elements of $Assurance\_Cert$ are discussed next:

1. $Cert_{ID}$: The unique identifier of the certificate.
2. $Dev_{ID}$: The unique identity information of the `IoT Device` receiving the $Assurance\_Cert$.
3. $device\_profile$: As stated previously, $device\_profile$ identifies the usage context of the `IoT Device`. It is used by the CAB to perform profile-specific risk evaluation and is sent to the CAB by `Device Vendor` requesting device certification.
4. $assurance\_level$: The assurance level of the `IoT Device` which can be in the form of a rating (`Basic, Substantial, High`), or a score (e.g. 0-10). This rating is used by `End-user` to define local access policies for different levels of security. It is pertinent to state that the health rating can be different in a subsequent evaluation of the same software stack due to emergence of new vulnerabilities and threats. This results in a short-lived $Assurance\_Cert$ validity.
5. $T$: Validity period of the certificate.
6. $CK_{pk}$: The key which will later be used to communicate securely with the certified `IoT Device`.

Further Discussions: Table 3.1 summarises how TruCerT protocol fulfills the set of requirements necessary for a broadly usable certification process, based on the EU Cybersecurity Certification Framework. A certified `IoT Device` is shipped with its $Assurance\_Cert$ to the `End User`. The included security assurance level allows the `End-user` to apply relevant security policies to grant access in the user network. To establish

*Table 3.1. Requirement fulfillment by TruCerT mechanisms*

| **Req.** | **Fulfillment** |
|---|---|
| Minimum baseline | TruCerT introduces device_profile, requirement of TPM-support and calculation of assurance level to establish a minimum baseline. |
| Minimum overhead | The communication overhead of the protocol is at par with RATS. However, estimation of overhead of the entire certification process requires further evaluations which are part of future work. |
| Lean reassessments | TruCerT maintains regularity as it enables re-assessment and re-certification automatically when the certificate is invalidated. |
| Risk assessment | TruCerT proposes mechanisms for risk assessment and evaluation in Section 3.2.4. It specifies the usage scenario and context of the IoT device as the device_profile to support better risk assessment. |
| Penetration testing | TruCerT does not discuss penetration testing. |
| Vuln. management | TruCerT protocol is limited to audit and certification mechanisms but describes on a high-level in Section 3.2.4 how the vulnerability management databases can be used with TruCerT. |
| Integration w/ standards | Section 3.3 discusses this requirement and its fulfillment in detail. |

a communication session with an IoT Device, the `End-user` requests the certificate, validates it and verifies the device.

## 3.3 Realization of TruCerT with Standards

In this section, we discuss TruCerT with respect to existing standards i.e. RATS for RIV of device, and X509 device authentication certificates. We show how TruCerT goals can be achieved using existing standards and regulations.

### 3.3.1 TruCerT with RATS

Remote Attestation Procedures (RATS) [39, 85], describes a process and guidelines for remote attestation of the integrity of firmware and software installed on networked devices that contain TPM1.2 or TPM2.0. RATS includes mechanisms for RIV processes like (i) device identification and (ii) generation, conveyance and appraisal of cryptographic proof of the device in question. As mentioned previously, TruCerT relies on verifying the current state of the IoT device's software stack for certification. Hence, RATS can be used instead of the proposed TruCerT mechanism using TPM-generated $CK$. We briefly discuss TruCerT accompanied with RATS for achieving RIV (Fig. 3.3 is the workflow of TruCerT with RATS). This procedure successfully verifies the integrity of the software-state but does not prevent the TOCTOU problem [42]. The $Assurance\_Cert$ received after RATS-based audit does not guarantee the same assurance level at the time of *use* of the certificate. Using TPM mechanisms to bind the certificate to the software-state, as proposed in TruCerT ensures that the certificate being used has the same assurance level that was audited and is stated in the $Assurance\_Cert$.

### 3.3.2 TruCerT and X509-based Certificates

Major IoT deployment models today support IoT device authentication based on X509 client certificates [128, 5]. These certificates consist of identity information of the device, the Certification Authority (CA), the certificate itself and some extensions. The

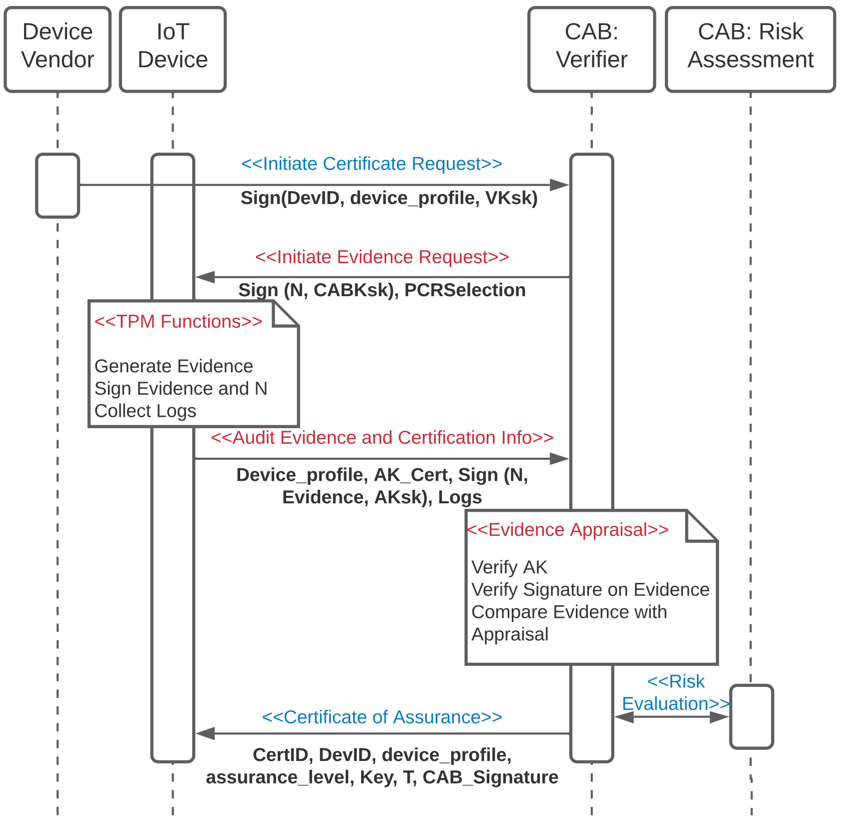

Device Vendor
IoT Device
CAB: Verifier
CAB: Risk Assessment
<<Initiate Certificate Request>>
Sign(DevID, device_profile, VKsk)
<<Initiate Evidence Request>>
Sign (N, CABKsk), PCRSelection
<<TPM Functions>>
Generate Evidence
Sign Evidence and N
Collect Logs
<<Audit Evidence and Certification Info>>
Device_profile, AK_Cert, Sign (N, Evidence, AKsk), Logs
<<Evidence Appraisal>>
Verify AK
Verify Signature on Evidence
Compare Evidence with Appraisal
<<Risk Evaluation>>
<<Certificate of Assurance>>
CertID, DevID, device_profile, assurance_level, Key, T, CAB_Signature

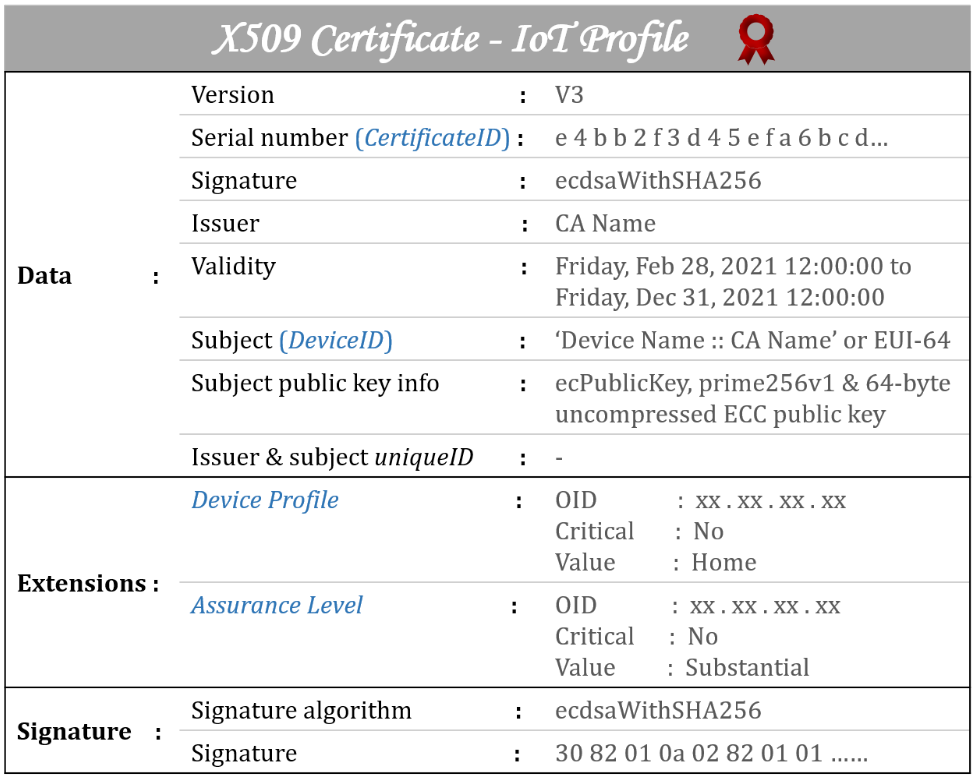
*Figure 3.3. TruCerT Protocol with RATS for Remote Integrity Verification*

*extensions* defined for X.509 certificates are a way to introduce additional attributes like users or public keys and manage relationships between CAs. Each *extensions* in a certificate is designated as either critical or non-critical.

TruCerT proposes to include the *Assurance Attributes* represented in $Assurance\_Cert$ using standard X509 certificates. In existing architecture, the IoT device receives a Device Identity Certificate (we call it *Cert*) through Est-CoAP from the CA. These certificates authenticate the device during client-server communications. We propose that after the CAB performs RIV of the IoT device, it transfers the *Assurance Attributes* specified in the $Assurance\_Cert$ to the CA. The CA then updates the *Cert* of the device with *Assurance Attributes* and enrolls a new certificate, i.e. *Cert + Assurance Attributes*. This would require a new certificate request, resulting in a new certificate serial number. The new Thumbprint/Fingerprint of the certificate will represent both the *Identity* and *Assurance Attributes* of the device. The IoT profile of standard X509 certificate is shown in figure 3.4, it shows how the attributes of $Assurance\_Cert$ can be accommodated in the certificate *extensions*. Figure 3.5 depicts the original TruCerT protocol and the alternate proposal extending the existing IoT architecture.

## 3.4 Network Overhead of TruCerT vs TruCerT with RATS

In this section, we discuss a deployment scenario for a TPM-based IoT device and assess the communication overhead between the IoT device and Verifier (CAB) for RIV. The network overhead of the communication in the original TruCerT protocol is then compared to the TruCerT version with RATS for RIV. We consider an IoT device of Raspberry Pi 4 [148] type supporting TPM 2.0. The device is part of an IoT network, connected to the manufacturer's server and the Verifier for audit and certification



X509 Certificate - IoT Profile

| | | | |
|---|---|---|---|
| Data : | Version | : | V3 |
| | Serial number *(CertificateID)* | : | e 4 b b 2 f 3 d 4 5 e f a 6 b c d... |
| | Signature | : | ecdsaWithSHA256 |
| | Issuer | : | CA Name |
| | Validity | : | Friday, Feb 28, 2021 12:00:00 to Friday, Dec 31, 2021 12:00:00 |
| | Subject *(DeviceID)* | : | 'Device Name :: CA Name' or EUI-64 |
| | Subject public key info | : | ecPublicKey, prime256v1 & 64-byte uncompressed ECC public key |
| | Issuer & subject *uniqueID* | : | - |
| Extensions : | *Device Profile* | : | OID : xx . xx . xx . xx<br>Critical : No<br>Value : Home |
| | *Assurance Level* | : | OID : xx . xx . xx . xx<br>Critical : No<br>Value : Substantial |
| Signature : | Signature algorithm | : | ecdsaWithSHA256 |
| | Signature | : | 30 82 01 0a 02 82 01 01 ...... |

*Figure 3.4. Standard X509 IoT profile with TruCerT assurance attributes: Certificate ID, Device ID, Device Profile and Assurance Level*

purposes. Most configurations of a TPM are implementation-specific, e.g, TPM 2.0 supports varying number of PCR(s). For our evaluation, we consider a TPM supporting 32 PCRs. The size of PCR depends on the algorithm used for extending it. We use SHA256 for generating all hashes in this analysis. The TPM signatures are performed using RSA with a 2048 bit key. The *proof_CK* and *Evidence* are TPM-generated values and are held in `TPM2B_ATTEST` structure. Standard RATS uses the YANG interface for challenge-response communication that implements the TCG TAP model [38]. Under this model, all communication to exchange attestation information takes place via YANG RPCs based on XML. The values of *PCRSelection* and *Evidence* follow the YANG model specifications. The communication in TruCerT and in TruCerT with RATS can be labelled with 5 steps. (i) Step 1: Initiate Certificate Request, (ii) Step 2: Initiate Evidence Request, (iii) Step 3: Audit Evidence and Certification Info, (iv) Step 4: Risk Evaluation, and (v) Step 5: Certificate of Assurance.

Steps 2 and 3 in TruCerT can be replaced with RATS, whereas Steps 1, 4 and 5 remain same in terms of communication overhead in both mechanisms. During Steps 2 and 3 of TruCerT, the following messages are exchanged:

- In Step 2, the Verifier sends a signed *N* of size *n*, generated using a recommended cryptographic-strength algorithm.
- The IoT device sends $IML_{dev}$ of size *i* and *device_profile* of size *d*. The certification key *CK* sent is of 256 bits. The IoT device computes proof of the key i.e. *proof_CK* of 272 bits and sends it along with the 2048 bit signature generated by TPM. The TPM certificate *AK_Cert* of size *a* is also sent.

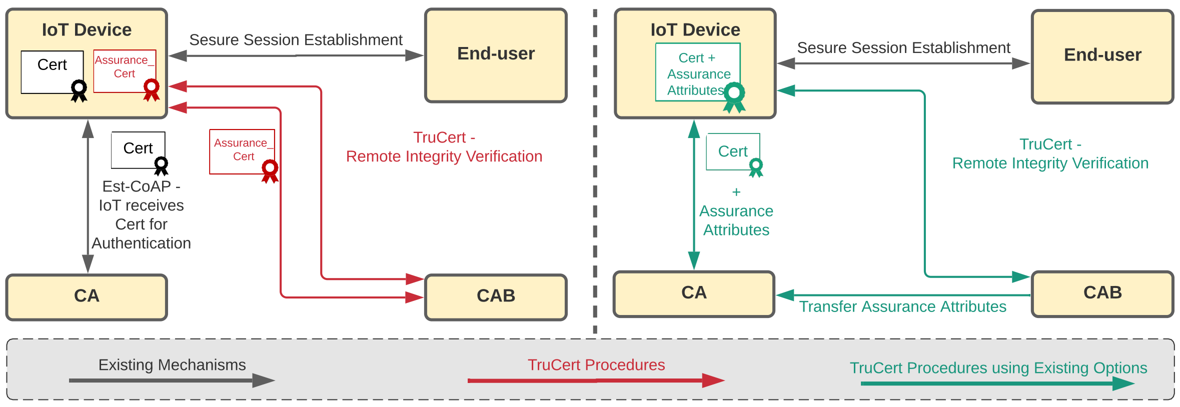

IoT Device
Cert
Assurance_Cert
Sesure Session Establishment
End-user
Cert
Est-CoAP - IoT receives Cert for Authentication
Assurance_Cert
TruCert - Remote Integrity Verification
CA
CAB
IoT Device
Cert + Assurance Attributes
Sesure Session Establishment
End-user
Cert
+ Assurance Attributes
TruCert - Remote Integrity Verification
CA
Transfer Assurance Attributes
CAB
Existing Mechanisms
TruCert Procedures
TruCert Procedures using Existing Options

*Figure 3.5. Introduction of TruCerT and Assurance_Cert into existing IoT paradigm.* `IoT Device` *receives a Device Identity Certificate, i.e.* Cert *through Est-CoAP from CA, which is used for secure session establishment with* `End-user`*.* `CAB` *audits the device using TruCerT and achieves RIV. The device then receives an* Assurance_Cert *from* `CAB` *which guarantees software-state integrity. Another proposal presented here is combining the assurance attributes with X509 Device Identity Certificates. TruCerT proposes this alternate mechanism to transfer the* Assurance Attributes *after RIV to the CA. The CA then updates the* Cert *of* `IoT Device` *with* Assurance Attributes *and enrolls a new certificate, i.e.* Cert + Assurance Attributes.

- Therefore, for one successful RIV of the device, the total communication overhead is: $n + i + d + 256 + 272 + 2048 + a$; where *n, i, d* and *a* are constant values in bits.

Messages exchanged in Step 2, 3 of TruCerT with RATS are:

- The Verifier sends a signed *N* of size *n* and a set of PCR indices i.e. *PCRSelection*. The indices are represented using YANG datatype `uint8` [94] and sending a single index would take 8 bits, whereas sending 32 indices would require 256 bits.
- The IoT device sends *Log* of the same size as $IML_{dev}$ i.e. size *i*, and a *device_profile* of size *d*. The TPM generates the *Evidence* of 272 bits using the requested PCR indices and *N* and sends it along with the 2048 bit signature. Finally, the IoT device also sends the TPM certificate *AK_Cert* of size *a* (as in TruCerT protocol).
- Hence, the communication overhead for TruCerT with RATS for RIV is: $n + 8 + i + d + 272 + 2048 + a$ for one PCR index and $n + 256 + i + d + 272 + 2048 + a$ for 32 indices selected; where *n, i, d* and *a* are constant values in bits.

TruCerT communication overhead is at par with TruCerT with RATS for RIV with a minor difference of 248 bits in worst case scenario. The size of *N*, *device_profile*, *Logs*, *AK_Cert* and *Signature* remain same across both versions of the protocol. The factors contributing to the difference in overhead are *PCRSelection* and *Evidence* for TruCerT with RATS and *CK* and *proof_CK* in TruCerT. With TruCerT, we generate assurance certificate and achieve automated certification without being affected by the TOCTOU problem which is our tradeoff for a few bytes of communication overhead.

## 3.5 Security Analysis

### 3.5.1 Introduction to Tamarin

Tamarin [122] is a symbolic analysis tool using multi-set *rewriting rules* —to encode a protocol specification and the adversary's capabilities— and *first-order logic formulas* —to define security properties. These rewriting rules induce a transition system describing the potential executions of (unbounded numbers of) protocol instances in parallel, allowing us to cover the potential interactions of enrolling at many CAs and using various services concurrently. Tamarin's default adversary model corresponds to a Dolev-Yao [62] adversary who has complete control over the network: it can eavesdrop, block, or modify messages sent by honest agents. It can also inject messages of its own, provided it knows all the necessary information to build them (e.g. to generate a given ciphertext, the adversary needs to know the associated private key). This model allows all sorts of man-in-the-middle, impersonation, reflection, or relay attacks, among others. Users can also extend this default model, either to give more capabilities to the adversary (e.g. learn random values through an oracle) or limit what it can do on the network (e.g. confidential channels that cannot be eavesdropped on).

### 3.5.2 TruCerT Formal Analysis

We modeled the communication between the `Device Vendor`, the `IoT Device`, and the `CAB` Verifier. The Verifier and Risk Assessment agents are generally the same entity and we separated them for ease of presentation and making the parallels between TruCerT and RATS clearer. In any case, as the communication between these two agents handle no cryptographically sensitive data, it doesn't need to be modeled. In addition to its full control of the network, we let the adversary compromise private keys, and show that our security properties hold for a given protocol instance as long as the secret keys involved in this instance are not compromised. Indeed, TruCerT crucially relies on the security properties provided by TPMs, and the `CAB` as a trusted third-party, and does not give any guarantee if one of them is compromised. As for the `Device Vendor`, the only attack it can perform if compromised is sending bogus certificate request in an attempt at Denial of Service, a class of attacks that is not covered by the Tamarin security model. Both versions, with or without RATS, have the same Tamarin model. We proved the secrecy of $CK_{sk}$, thus guaranteeing that certificates are not spoofable, and a strong authenticity property —injective agreement in both directions between the `CAB` and the `Device Vendor`, from Lowe's hierarchy of authentication specifications[117]— which guarantees that the certificate of assurance received by the `IoT Device` is correct and that no replay attack is possible and that the adversary cannot perform an attack by making several runs of TruCerT interact. Our model is available at `https://github.com/Simon-Bouget/tamarin-TruCerT/blob/main/TruCerT.spthy` and can be automatically verified with Tamarin's default heuristic in under two minutes.

# 4 Secure computing with Contiki-NG

In this chapter, we describe our implementation efforts to provide the IoT ecosystem with a secure software stack where all critical security building blocks are readily available. Our efforts focused on two different layers, each presented in the next two sections.

## 4.1 Contiki-NG on Low-Power RISC-V Devices

### 4.1.1 Background knowledge and context of the work

We need a framework for work with Cybersecure IoT platforms based on RISC-V. Since there are some open designs of RISC-V cores that can run on FPGAs, the focus is on setting up a Contiki-NG platform for one of the open designs for FPGAs and, from that platform, extend with security features over time.

#### 4.1.1.1 Description of RISC-V CPU architecture (SERV)

To develop a small RISC-V Contiki-NG port, we selected a small implementation of a 32-bit RISC-V core by Olof Kindgren called SERV[1], the world's smallest RISC-V core. Based on SERV, there is a reference platform for FPGAs called Servant, a minimal SoC with a UART and timer, illustrated in figure 4.1. Servant supports Zephyr RTOS. Servant can be run in verilator for testing and evaluating the Contiki-NG port.

#### 4.1.1.2 RISC-V on FPGA

The target for the porting effort is to have the ability to research security on the hardware side and the software side, using FPGAs for faster simulation of the full system (e.g. SoC design and software) to support the technology exploration and build RISC-V experience. The construction of a RISC-V testbed based on FPGA for prototyping of open-source RISC-V processors and platforms, provides an environment in which new platforms and ideas can be tested out without the need for continuous investment in new physical hardware (which lags the scientific state of the art because of the complex and expensive process of tape out/manufacturing).

The FPGA prototyping testbed used is a Xilinx VCU128 FPGA development kit, which is a large FPGA specifically intended for hardware prototyping. It consists of 2 852k logic cells, 9024 DSP slices, and 340,9 MB of block RAM. For external communication it has 624 HP I/O, 8 GB of HBM, and several peripheral interfaces. Besides target platform for the SERV architecture, which is the focus of this report, the VCU128 is capable enough to simulate larger systems, and there is ongoing work to develop the software stack for the Carfield project on VCU128 FPGA boards. That project is in itself also an interesting testbed for exploration of Root of Trust (RoT) on RISC-V, as such components are included in the soft-core platform, and a promising future step for these efforts.

### 4.1.2 The Contiki-NG OS

Contiki-NG (Next Generation) is an open-source, cross-platform operating system for severely constrained wireless embedded devices. It focuses on dependable (reliable

[1] `https://github.com/olofk/serv`

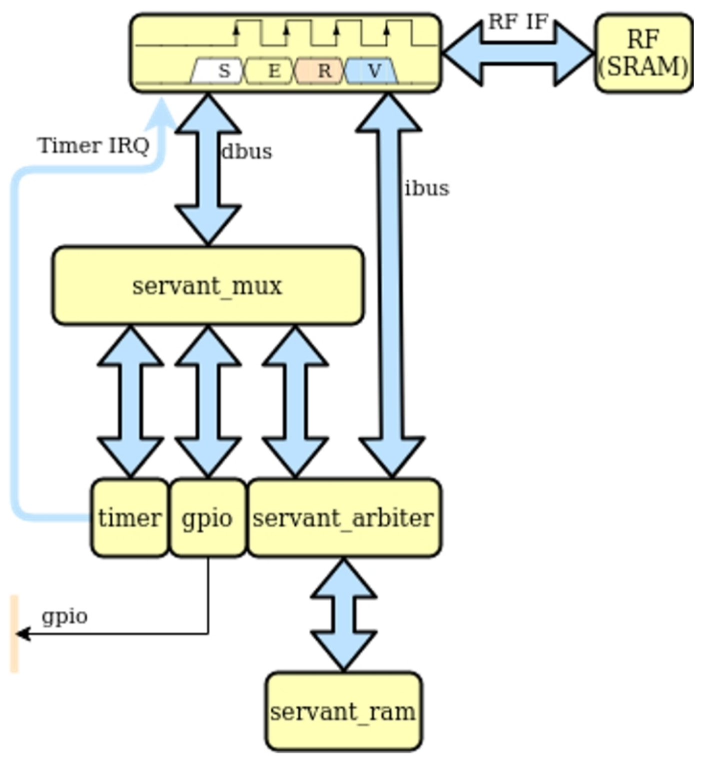


*Figure 4.1. Servant with SERV (from Olof Kindgrens github repository)*

and secure) low-power communications and standardized protocols, such as 6LoWPAN, IPv6, 6TiSCH, RPL, and CoAP. Its primary aims are to (i) facilitate rapid prototyping and evaluation of Internet of Things research ideas, (ii) reduce time-to-market for Internet of Things applications, and (iii) provide an easy-to-use platform for teaching embedded systems-related courses in higher education. Contiki-NG supports Trust Zone for ARM SoCs and will add support for similar trusted execution environments for RISC-V in a later phase.

### 4.1.3 Porting Contiki-NG to a RISC-V CPU

The Contiki-NG port to SERV is a first step for enabling experimentation of hardware/-software security features within the Contiki-NG ecosystem. Since there is an initial port for Zephyr OS[2], we have made use of the Zephyr SDK (including the tools needed to build RISC-V) as a starting point. The port follows the Contiki-NG guide for new platforms: it makes Contiki-NG capable of running on the “bare-bone” hardware without any other external software components. Hence it can be called a low-level ”native” Contiki-NG port. Contiki-NG build system is based on GNU Make. The serv/RISC-V port adopts the elements of serv’s port of Zephyr OS. In particular, the following parts are used:

- The software development kit’s elements – including the target-specific compiler and linker
- The library environment and building options
- Source files implementing hardware initialization and basic services (e.g. timer)

[2] `https://zephyrproject.org`

File structure of the port:

- contiki-ng/arch/cpu/serv/... - This is the SERV MCU part of the port. Defines compilers, linkers, etc for the SERV target RISC-V microprocessor.
- contiki-ng/arch/platform/servant/... - This is the SoC or servant part where the support for the complete SoC including peripherals such as UART is included.

The current version is available on a fork of Contiki-NG at `https://github.com/joakimeriksson/contiki-ng`, but long term our goal is to have this implementation available in the official Contiki-NG main repo.

#### 4.1.4 Future Work

The selected SoC, SERV / Servant, does not support any security features. Still, for RISC-V, there are several open designs on Trust Zone-like security mechanisms that enable a trusted execution environment (TEE). Some examples are:

- MultiZone that enables TEE for RISC-V designs. MultiZone enables multiple equally secure domains, with full control over data, programs and peripherals. More details at: `https://hex-five.com/multizone-security-tee-riscv/`, and source at `https://github.com/hex-five/multizone-sdk`
- SiFive WorldGuard – another TEE for RISC-V designed by SiFive and donated to the RISC-V international (the RISC-V organization/alliance). WorldGuard like MultiZone do support multiple secure domains (or worlds). More details: `https://www.businesswire.com/news/home/20230524005055/en/SiFive-Gives-WorldGuard-to-RISC-V-International-to-Make-this-Robust-Security-Model-More-Accessible-to-the-RISC-V-Community`

The focus of future work for the Contiki-NG port is to find a TEE that is well suited to IoT platforms that typically are relatively resource-constrained but very connected and, therefore, will need a lightweight but very secure TEE.

In parallel with the aforementioned ”native” Contiki-NG port to the target RISC-V platform (serv), we are also working on implementing a new CMake-based build system instead of – or in addition to - the existing one based on GNU Make. This would allow:

- Factorize Contiki-NG implementation into individual sub-components, in particular – elements of the networking stack, and be able to use them independently of the low-level ”native” hardware Contiki-NG dependencies.
- Build Contiki-NG applications as threads in other RTOSes built around CMake - in our case, Zephyr OS. That is, the functionality of Contiki-NG becomes available for use within Zephyr OS SDK ”ecosystem”, including Zephyr’s existing and well-tested ports to RISC-V CPUs/platforms, including ‘serv’

## 4.2 EDHOC for Contiki-NG

We are implementing the newly standardized lightweight authenticated key exchange called EDHOC (Ephemeral Diffie-Hellman Over COSE) standard for the Contiki-NG.

EDHOC is a de facto standard for key management for low-power IoT devices, primarily developed for OSCORE, a new cryptographic standard for message security.

RISE worked on implementing the latest revision of Ephemeral Diffie-Hellman Over COSE (EDHOC) in Contiki-NG. This will add an essential component to the implementation of Object Security for Constrained RESTful Environments (OSCORE) in Contiki-NG.

The current status is that the majority of a basic implementation is implemented along with an automated testbench. The work is being validated against section 3 in draft-ietf-lake-traces-08.

The code is available at `https://github.com/pjonsson/contiki-ng/tree/edhoc-drop`

The work has also resulted in user interface improvements in Cooja and build system fixes in Contiki-NG getting merged upstream and that will be a part of the Contiki-NG 5.0 release.

# 5 SIRE Evaluation

In this chapter, we consider the application of SIRE (acronym for truSted verIfieR sErvice) in three different use cases: for IoT membership management, for autonomous vehicle coordination, and for robust federated machine learning.

We note that the need for membership management and attestation of IoT devices is highlighted in a demonstration that has been prepared in VEDLIoT, in the scope of the motor monitor use case. In this demonstration, which also brings together the secure MQTT broker described in Section 2.3, SIRE is used for device attestation to illustrate how the service is used. The other use case scenarios were considered both because they are relevant and related to other work in VEDLIoT, namely to the pedestrian detection in autonomous driving use case and to the federated learning use case considered in the FLAIR Open Call project, and because these use case scenarios may involve a large number of IoT devices.

The description of SIRE and its implementation was provided in Deliverables D5.2 and D5.3. Therefore, here we directly delve into the description of the use cases, followed by evaluation results concerning the performance of SIRE in each of them.

## 5.1 Use Cases

To better illustrate and motivate the need for a solution that combines remote attestation and coordination primitives, we will detail a few use cases and environments in which the employment of a solution like SIRE could be a suitable approach.

### 5.1.1 IoT Membership Management

The membership of a system is represented by an up-to-date list of active participants, which is an especially complicated task in some IoT applications due to the scale and dynamism of the device group. In cloud-based systems, membership management is performed using a coordination service due to two fundamental characteristics: failure detection of client processes and highly available consistent storage [61]. Typically, a node that wants to join the system will request the coordination service to add it to the application's membership without any concern for its correctness. Although this might be appropriate in a data center, in an IoT scenario, accepting every device that requests to join an application can constitute a considerable threat as these environments are unsafe. Therefore, the coordination service must support the definition of policies that specify the requirements for participating devices to ensure a certain degree of security.

### 5.1.2 Autonomous Vehicle Coordination

It is expected that in the near future, we will have our streets filled with autonomous self-driving vehicles, which will require coordination of their movements. Such coordination enables proper access to shared resources, such as intersections and parking slots, and the execution of mobility tasks, such as platooning and ramp merging [119]. Some of these tasks, such as parking slot management, typically use a centralized infrastructure to maintain the state of both the environment and the participants (e.g., keep a list of the slots in a smart parking lot and their occupation state as well as the state of the occupying vehicles [99]). Other tasks, such as platooning, rely primarily on inter-vehicle communication to execute the decision-making necessary to perform the task. In addition, there are also some tasks, such as intersection management, which can be performed in both ways depending on the available resources (e.g., in the case an intersection does not have a smart traffic light, the order of crossing is decided between the vehicles [195]). For the purpose of this paper, we opted to focus on the intersection management use case.

#### 5.1.2.1 Intersection Management

Whenever multiple vehicles meet at an intersection, a unanimous order of passage needs to be established to prevent traffic accidents from occurring. This decision can be taken either by a centralized infrastructure, i.e., a ”smart” traffic light, as pictured in Figure 5.1, or cooperatively by all the conflicting vehicles. Some systems utilize the ”smart” traffic light by default and rely on the latter approach only when the infrastructure is absent [195].

Although these tasks can be reduced to a matter of coordination, some characteristics and issues inherent to the autonomous vehicle environment make existent coordination services unsuited for the field. First, this environment requires a safety-critical design strategy as even minor faults could have high costs, both in the human and monetary sense. Thus, assuring the correctness of every entity that composes the system is essential. As an exception to the IoT field, the autonomous vehicle environment is fairly homogeneous, with smart vehicles having high computation capabilities as they are required to perform heavy computational tasks, such as machine learning. Although this might be true for the current scenario, it is fair to expect that, with the prevalence of smart vehicles, communications will need to be performed with other road entities, such as pedestrians and cyclists. Therefore, the field might quickly become heterogeneous if these entities are taken into account, and as such, it is beneficial to develop solutions adapted to client heterogeneity. Lastly, these environments are composed of a large quantity of highly dynamic participants, making it difficult to keep an up-to-date view of the system's membership and state. Although some systems and tasks do not require it, there is always a benefit in having an up-to-date and centralized source of information available to joining vehicles, for example, or even to facilitate enabling the execution of other tasks that are not available in the system.

### 5.1.3 Robust Federated Machine Learning

Machine learning models have grown in complexity and in the amount of data processed, which requires a lot of computation resources. Therefore, most machine learning implementations are now distributed. Most of these implementations rely on a core component, a parameter server, which is in charge of updating the parameter vector, while workers perform the actual update estimation based on the share of data they have access to. This data could be a fraction of a dataset provided by the parameter server or even data directly gathered by these workers. These workers may be devices of any kind, from simple sensors and cameras to hardware accelerators or even regular computers. This scenario is described in Figure 5.1. Although the implementations are distributed, none can tolerate computation errors, stalled processes, or attackers trying to compromise the system. One approach to solve this problem is to use an aggregation rule that can tolerate Byzantine computation faults, such as

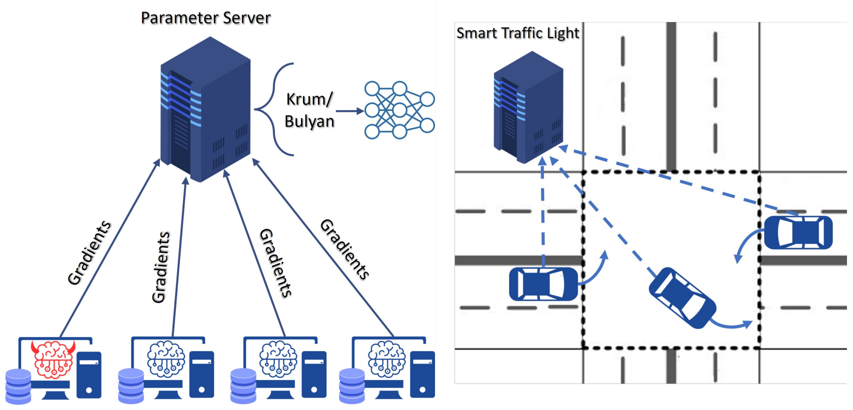


*Figure 5.1. SIRE Use Cases: BFT federated learning and autonomous vehicle intersection management.*

Krum/Multi-Krum [40] and Bulyan [127]. Two examples of systems that implement and utilize these rules are AggregaThor [55] and Garfield [84]. Essentially, these rules take all the gradients computed by the workers as input and discard those that differ greatly from the others, meaning that these rules are only effective if most workers are correct.

Similarly to the autonomous vehicle environment, the parameter server scenario has a few characteristics and issues that should be considered when developing appropriate solutions. First, this scenario can be heterogeneous as the workers may be devices of any kind with a wide range of computational capabilities. Although the Byzantine aggregation rule can prevent malicious workers from impacting the parameter vector, these adversaries can still interact with the parameter server and, as such, may constitute a threat to the system. Therefore, some guarantees must be given regarding the correctness and integrity of the workers. Lastly, in some systems, the aggregation rule is executed when the parameter server has collected gradients from each worker. This requires the utilization of failure detection to prevent crashed workers from stalling the entire system.

## 5.2 Evaluation

In this section, we present the results of SIRE's performance evaluation. These experiments consist of the typical distributed systems' read and write (i.e., $get$ and $put$ operations) experiments followed by an evaluation of SIRE's performance on the three use cases presented in Section 5.1: IoT Membership Management ($join/attest$ operation), autonomous vehicle intersection management, and robust federated learning.

*Experimental setup:* All tests were executed in 16 machines with the following characteristics: Dell PowerEdge R410 with two quad-core Intel Xeon E5520 (2.27 GHz) CPUs or two quad-core Intel Xeon E5620 (2.40 GHz); two hardware threads per core; 32 GB of RAM; gigabit ethernet connection; Ubuntu 22.04 operating system with OpenJDK RE 11.0.19. The five machines with the more powerful CPU, Xeon E5620, were always used to execute the server replicas. In each experiment, we utilized up to 2,000 clients distributed uniformly across 10 machines. Since the cluster only has 16 machines, we had to utilize additional machines with different specifications when running experiments with seven and ten replicas.

### 5.2.1 Read and Write Data – *get* & *put*

We start by reporting the results of read and write performance tests that are commonly used to evaluate distributed systems. These tests were performed with four, seven, and ten server replicas to tolerate one, two, and three faults (represented by $f$), respectively, with each replica being executed in its own machine.[1] We deployed up to 2000 clients across multiple machines, each running its own proxy to impose a higher load on the server replicas. One additional machine was used to gather the measurements without executing any operations to prevent the measuring process from affecting the results.

We performed this evaluation using rounds, with the number of clients increasing between them, from 1 to 2000 clients, sending operations of 100 bytes. These clients were distributed equally between each machine.

In each round, the throughput measurements are taken around every two seconds by the leader replica that counts the number of requests in that period. This number is divided by two to obtain the throughput and stored, updating the max throughput accordingly. For the latency, we measure it for each request in the client, store it, and update the max latency accordingly. The results of this evaluation are presented in Figure 5.2 and Figure 5.3 for the $get$ and $put$ operations, respectively. Since the $get$ operation is executed unordered, i.e., it does not require the consensus protocol to be executed, its overall performance is higher than the $put$, as the latter must be executed in order. We can also notice that a higher number of server replicas (which implies a higher $f$) results in a lower throughput for both operations. The reason for this is the number of messages that must be exchanged between the replicas and the number of messages that must be sent to a client, as they are proportionate to the number of replicas. This leads to an especially increased load on the network connection, which cannot provide enough bandwidth to accommodate the large quantity of messages exchanged. However, in the case of the $get$ operation, the latency decreases with higher numbers of server replicas. Since it is an unordered operation, clients only need to receive responses from a fraction of the replicas ($2f + 1$) [37]. Hence, slower replicas impact the system's performance less, as the faster ones can compensate for the slower responses, leading to a lower latency. Lastly, it is relevant to mention that the measurements obtained in this experiment are very similar to the ones obtained in the performance evaluation of BFT-SMaRt by itself.

### 5.2.2 Use Case 1 - IoT Membership Management

The next experiment concerns the IoT membership management application. This use case comprises two major operations: $join/attest$ and $leave$. As the $leave$ operation is very similar in complexity to the $put$ operation evaluated in the first experiment, we will focus on the $join/attest$. This operation requires two roundtrips and must be executed ordered. Additionally, it employs an expensive signature algorithm (Schnorr [159, 160]), making it significantly more computationally complex.

We will follow the same experiment outline described for the previous experiment, measuring throughput and latency for a variable number of replicas and clients. The results of this experiment are presented in Figure 5.4. As we can see, the performance obtained is significantly lower than the one obtained for the $get$ and $put$ experiment.

[1]The number of replicas is given by the formula $n = 3f + 1$, where $n$ is the number of replicas and $f$ is the number of tolerated faults.

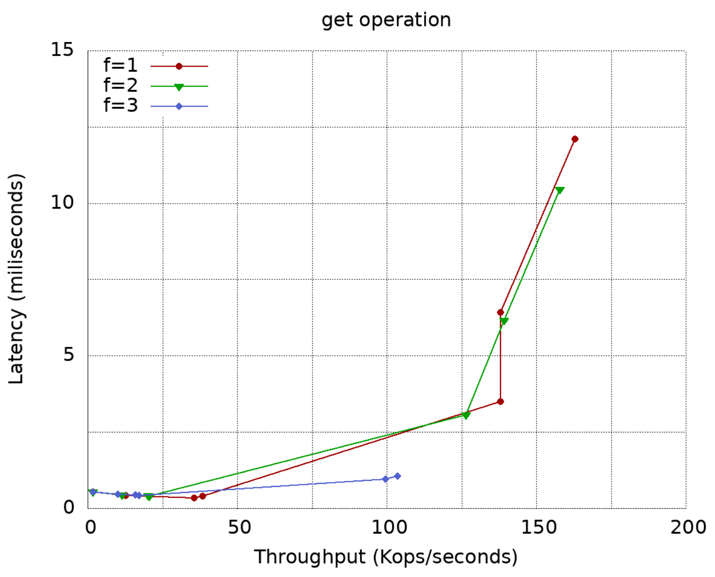


*Figure 5.2. Performance evaluation of the $get$ operation using a variable number of server replicas and a maximum of 2000 clients. $f$ represents the number of tolerated faults.*

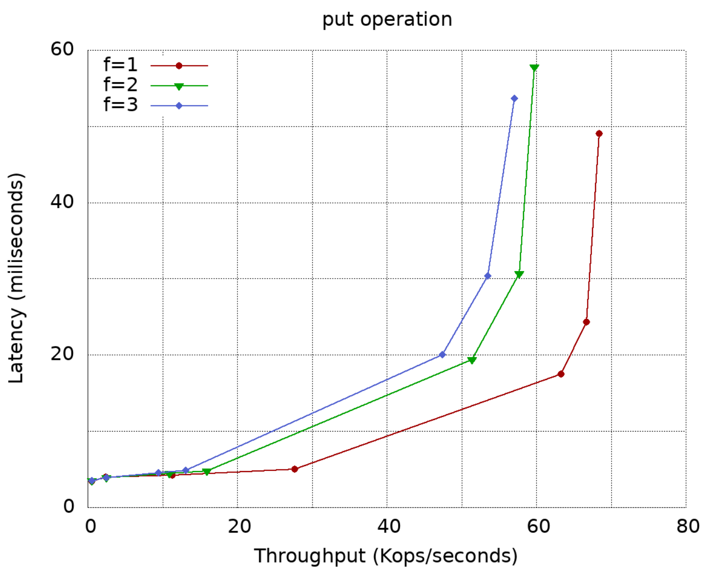


*Figure 5.3. Performance evaluation of the $put$ operation with a variable number of server replicas and a maximum of 2000 clients. $f$ represents the number of tolerated faults.*

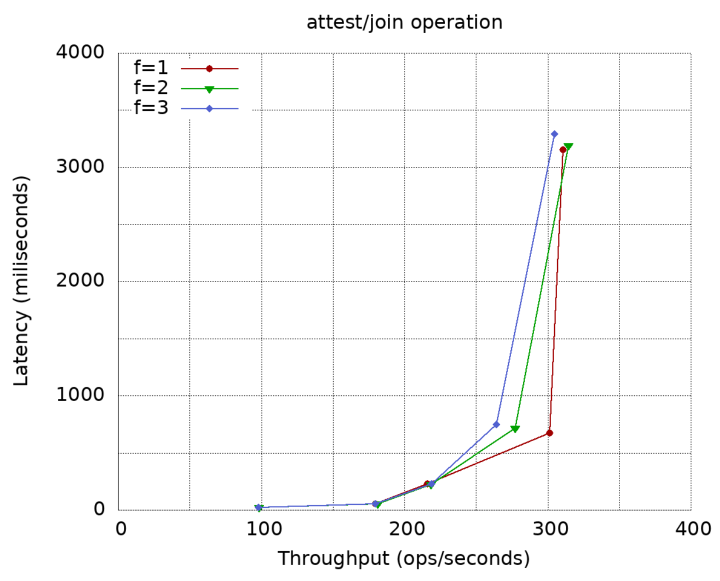


*Figure 5.4. Performance evaluation of the* $attest/join$ *operation with a variable number of server replicas and a maximum of 2000 clients.*

This is mainly because the Schnorr signature [159, 160] verification is very computationally heavy, which imposes a bottleneck on the attestation protocol.

Despite these performance values being significantly lower than the ones obtained for the $get$ and $put$ operations, these are still satisfying results given that the attestation protocol should only be executed ever so often and not something devices will be executing constantly, like the $get$ and $put$ Lastly, we would have liked to show a comparison in performance between our attestation protocol and the existing services. However, most literature on the topic is focused on the security aspect of the protocol and, therefore, does not present any performance metrics.

### 5.2.3 Use Case 2 - Intersection Management

For the intersection management use case, we adapted the algorithm from [195] as an extension for the coordination module. This solution assumes a four-way intersection with eight lanes where vehicles communicate their intention to a "smart traffic light," i.e., which lane they want to turn to. This smart traffic light will verify if that lane is "free" and return its decision to the vehicle. If the lane is free, the smart traffic light will lock three other lanes while the vehicle is crossing, while if the lane is locked, the vehicle will have to wait for the smart traffic light authorization. When a vehicle finishes crossing the intersection, it informs the smart traffic light that it is out of the intersection and, therefore, the previously locked lanes are now free. When the smart traffic light frees any lanes, it will check if any vehicles are waiting for their turn to cross and repeat the algorithm.

Since we are not utilizing actual vehicles, we simulate the crossing by setting a timer on vehicles between the authorization for crossing and the finish of the maneuver. We utilized a three-second timer in the scenario where a vehicle can cross immediately and five seconds in the scenario where a vehicle has to stop. Like in the previous experiments, we also performed multiple rounds. In this case, the rounds changed the number of vehicles/clients per hour (i.e., how many operations were sent per second) instead of the overall number of clients. We made this change to perform

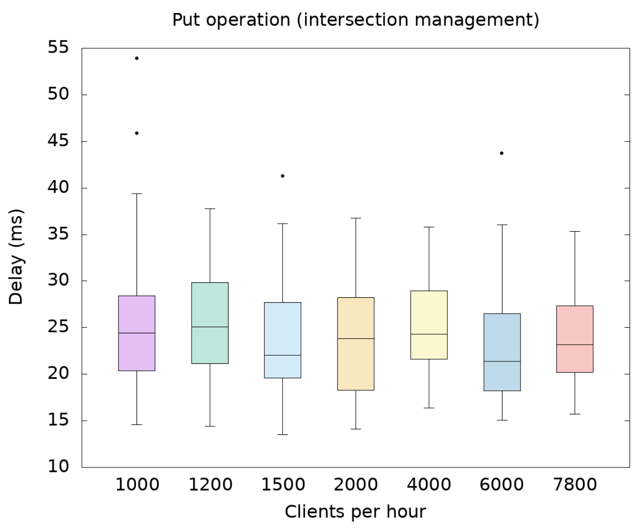


*Figure 5.5. Performance evaluation of the* $put$ *operation in the intersection management use case.*

tests more similar to the ones utilized in other intersection management systems. Additionally, we employed the same number of vehicles per hour commonly found in evaluating these types of systems, 1000 to 7800 vehicles per hour. Since the number of operations is significantly lower than the ones employed in the previous experiments, measuring throughput is not an adequate metric. Additionally, crossing an intersection takes three to five seconds, meaning the latency can never be lower than three seconds. Since our aim is not to evaluate the efficiency of the intersection management algorithm, we opted to perform measurements on the delay SIRE introduced on its execution. The measured delay is presented in Figure 5.5. We obtained an average of 3.5ms of added latency when utilizing SIRE to perform this algorithm. The obtained time includes the execution of the consensus protocol, the execution of the extension, and network transmission. With such a minimal delay, we can conclude that SIRE is suitable for this use case as it achieves similar results to a dedicated service while providing more functionalities and robustness guarantees.

### 5.2.4 Use Case 3 - BFT Federated Learning

Lastly, we evaluated SIRE as a parameter server in the BFT federated learning use case. We followed a similar algorithm to the ones presented in Garfield [84] and Agreggathor [55], i.e., a client/worker executes a training round, sends the gradients to the parameter server, which will execute an aggregation rule and return the resulting model. For this purpose, we adapted the Krum aggregation rule presented in [40] to be executed as an extension of the coordination module.

Contrary to the previous experiments, a Python client was utilized as machine learning methods are more easily implemented with this language. These clients utilized some pre-generated random data to perform their training using linear regression with gradient descent optimization. Machine learning tasks require large amounts of computational power, hence why this experiment utilizes significantly fewer clients than the previous (from 5 to 200 clients). Since this experiment aimed to evaluate the performance of SIRE in this use case, we did not perform any measurements regarding the accuracy of the obtained model. Instead, we followed a similar idea to the previous experiment and measured the delay SIRE introduced to the execution of

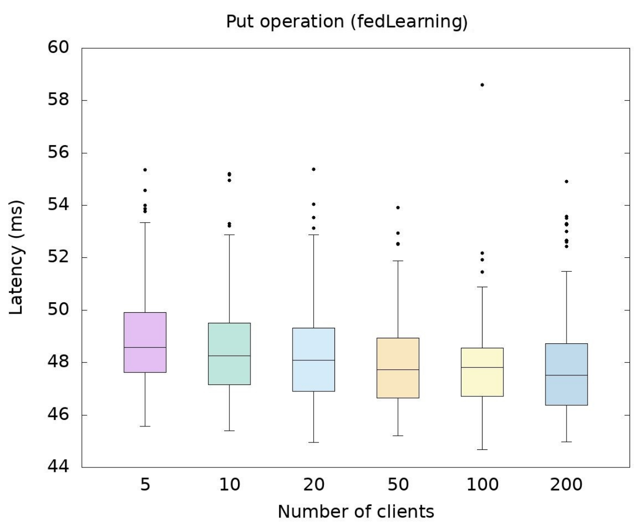


*Figure 5.6. Performance evaluation of the $put$ operation in the federated learning use case.*

the aggregation rule. The results of this experiment are presented in Figure 5.6. We obtained an average of 48.03ms of added latency when utilizing SIRE as a parameter server. The obtained time includes the execution of the consensus protocol, the execution of the aggregation rule, and network transmission. As you may notice, this delay is considerably higher than the one introduced in the intersection management. This is due to the larger messages being transmitted in this scenario and the increased complexity of the extension executed. Nonetheless, this delay is still within the acceptable range for the federated learning use case. Therefore, we can conclude that SIRE is suited for performing the parameter server role in this scenario as it achieves similar results to a dedicated service while providing more functionalities and robustness guarantees.

# 6 Simulation and Testing with Renode

During the course of VEDLIoT, as part of tasks 5.3 and 5.4, our focus was on establishing a framework for testing of accelerated, FPGA-oriented ML workflows in a simulated environment.

These tasks were centered around Renode [13] - Antmicro's open source simulation framework.

The entire scope of Tasks 5.3 and 5.4 can be summarized in the following points:

- Improving support for generated RISC-V-based SoCs
- Supporting Custom Function Units
- Improving general co-simulation capabilities of Renode
- Enabling automated testing of complex, generated platforms with co-simulated components

The work performed so far was thoroughly described as part of Deliverables D5.1 [145], D5.2 [100] and D5.3 [202].

The most recent work focused mainly on bringing together various improvements and features in order to present users with a comprehensive and coherent testing and development infrastructure based on Renode.

## 6.1 Introduction to Renode

Renode is Antmicro's open source simulation framework designed for functional simulation of embedded and IoT systems. It is based on building blocks and can simulate full Systems-on-Chip, also with multicore support, complete boards, as well as multinode systems. Renode features support for numerous architectures, currently including ARM Cortex-M, Cortex-R, Cortex-A, RISC-V, PowerPC, SPARC, Xtensa, and more.

Thanks to Renode's high level of accuracy in recreating actual hardware, the simulated systems bear high resemblance to their real life counterparts – it is possible to run the same exact software in simulation as one would on hardware, making the framework truly software-agnostic.

In the area of Machine Learning, being a central point of the VEDLIoT project, it's quite common to work with hardware that is still under development – either full systems or specialized accelerators. More and more companies are working on their software in parallel to their hardware, making mixed simulation scenarios, where part of the system is given but the accelerator part is in progress. This includes systems that are either supposed to be used in FPGA or are meant to be manufactured as ASICs. Renode's co-simulation capabilities let developers include in their simulated systems peripheral models that are written in Verilog [87], one of the popular Hardware Description Languages (HDLs), bringing such scenarios to life. These capabilities, compatible with HDL simulation tools like the open source Verilator [183] and proprietary Questa [163]

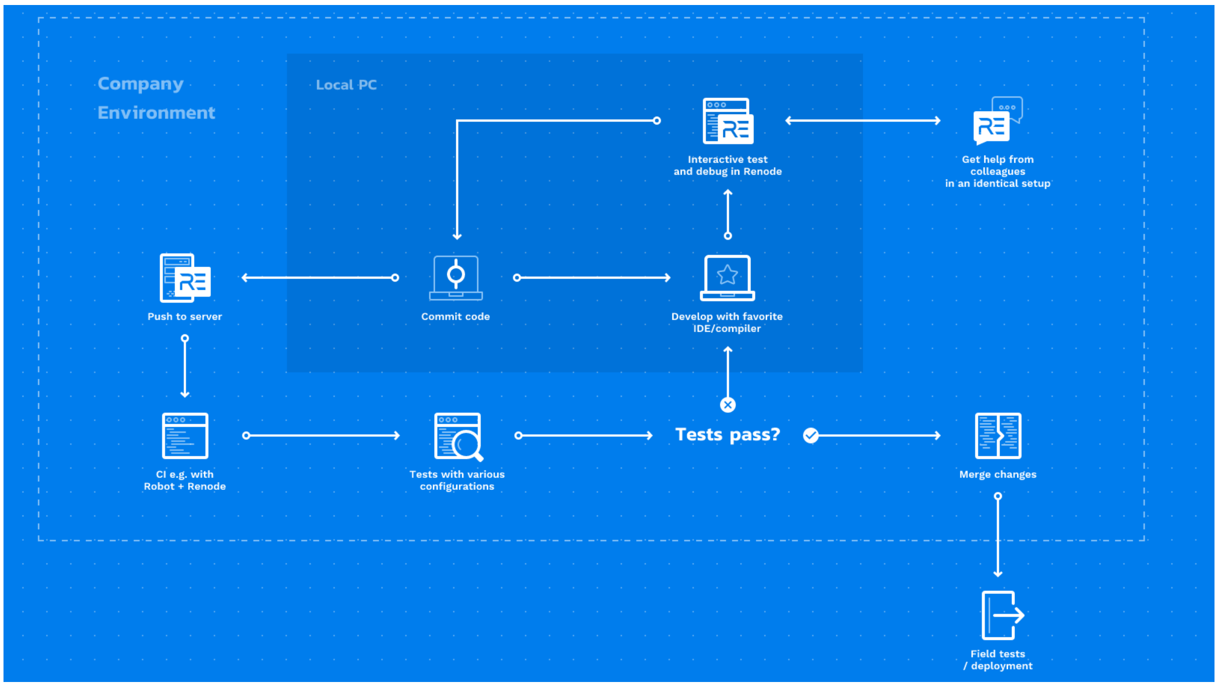

Company Environment
Local PC
Interactive test and debug in Renode
Get help from colleagues in an identical setup
Push to server
Commit code
Develop with favorite IDE/compiler
CI e.g. with Robot + Renode
Tests with various configurations
Tests pass?
Merge changes
Field tests / deployment

*Figure 6.1. CI-based Renode development flow diagram*

or Vivado [196], are explored in more detail below and in Deliverables D5.2 [100] and D5.3 [202].

Renode comes with several integrations with testing tools and libraries, which makes the framework useful for automated testing, e.g. in a Continuous Integration environment, where each code submission is automatically verified against a set of tests. The simulator is integrated with the open source Robot Framework project [69], a Python-based test automation and robotic process automation (RPA) tool, that can help you build complex, real-world test suites for embedded devices, and provides commands for running such tests. Renode is also integrated with Zephyr RTOS [170] Twister [171] (or Test Runner) via the Renode-Robot flow, as well as with Unity [172], a library for writing unit tests for C applications (and also integrated with Zephyr).

Simulation in Renode is deterministic which lets developers precisely reproduce simulated conditions every single time, resulting in reliable and replicable results, as opposed to testing on physical hardware, where certain conditions are extremely difficult to control, especially at scale.

All of these features, combined with extensive introspection capabilities allowing developers to extract detailed data from every software run, allow developers to set up CI-based development flows inside their organizations, similar to the flow presented in Fig 6.1, including Renode both in their local development work, but also running their collections of tests on remote systems.

## 6.2 Robust Automatic Platform Generation Features

One of the goals of our work was to enable developers to work on software targeting a platform created as part of the Task 4.7. As a result of automatic SoC generation, the platform is created from composable building blocks and can be loaded onto an FPGA platform.

Renode is able to use data created by the SoC generator in JSON format to create a Renode Platform description file (REPL) [14].

The concept of automatic platform generation was explored in other areas as well, both in the context of VEDLIoT and outside of the project's scope.

Renode is now able, using additional scripting, to analyze machine-readable platform descriptions in the following formats:

- JSON, created by the SoC generator used in Task 4.7
- HJSON, generated from the OpenTitan [141] build system
- Device Tree, used in Zephyr, U-Boot and Linux
- CMSIS-SVD, used in Renode mainly for logging purposes

All of these formats are used to describe Renode platforms in production environments.

JSON is used as part of the CFU Playground [78], a Google project that was the axis for introducing Custom Function Unit [8] support in Renode. A sample JSON file can look as follows:

```
{
    "csr_bases": {
        "ctrl": 4026531840,
        "ddrphy": 4026533888,
        "uart": 4026535936,
        [...]
    },
    "csr_registers": {
        "ctrl_reset": {
            "addr": 4026531840, "size": 1, "type": "rw"
        },
        [...]
        "uart_rxtx": {
            "addr": 4026535936, "size": 1, "type": "rw"
        },
        "uart_txfull": {
            "addr": 4026535940, "size": 1, "type": "ro"
        },
        [...]
    },
    "memories": {
        "clint": {
            "base": 4026597376, "size": 65536, "type": "io"
        },
        [...]
        "sram": {
            "base": 268435456, "size": 6144, "type": "cached"
        },
        [...]
    }
}
```

A REPL platform description generated from this input would be similar to:

```
rom: Memory.MappedMemory @ sysbus 0x0
    size: 0x10000

sram: Memory.MappedMemory @ sysbus 0x10000000
    size: 0x2000

clint: IRQControllers.CoreLevelInterruptor @ sysbus 0xf0010000
    frequency: 100000000
    numberOfTargets: 4
    [0, 1] -> cpu0@[101, 100]
    [2, 3] -> cpu1@[101, 100]
    [4, 5] -> cpu2@[101, 100]
    [6, 7] -> cpu3@[101, 100]

cpu0: CPU.VexRiscv @ sysbus
    cpuType: "rv32ima"
    privilegeArchitecture: PrivilegeArchitecture.Priv1_10
    hartId: 0
    timeProvider: clint

ctrl: Miscellaneous.LiteX_SoC_Controller_CSR32 @ { sysbus 0xf0000000 }

uart: UART.LiteX_UART @ { sysbus 0xf0001000 }
    -> plic@0

[...]
```

HJSON is used by the OpenTitan project [141] - an open source root of trust reference design, which is also well supported in Renode. This support was heavily developed and used within another Google project - Open Se Cura [12]. The goal of Open Se Cura is to provide an open source framework to accelerate the development of secure, scalable, transparent and efficient AI systems - a goal that is close in spirit to the goals of VEDLIoT.

Another recent mention-worthy development in the space of platform generation for Renode revolves around Device Trees [113]. Device Trees are used by a range of open source operating systems such as Zephyr or Linux, or bootloaders such as U-Boot, to describe platforms in a machine-readable way.

Some systems, like Linux, use device tree data in runtime to select appropriate drivers to initialize relevant devices. In other cases, like in Zephyr, a Device Tree Blob (DTB) is used in compile time to select drivers to be included in the final binary, thus reducing the final payload size and adding additional control during the build procedure.

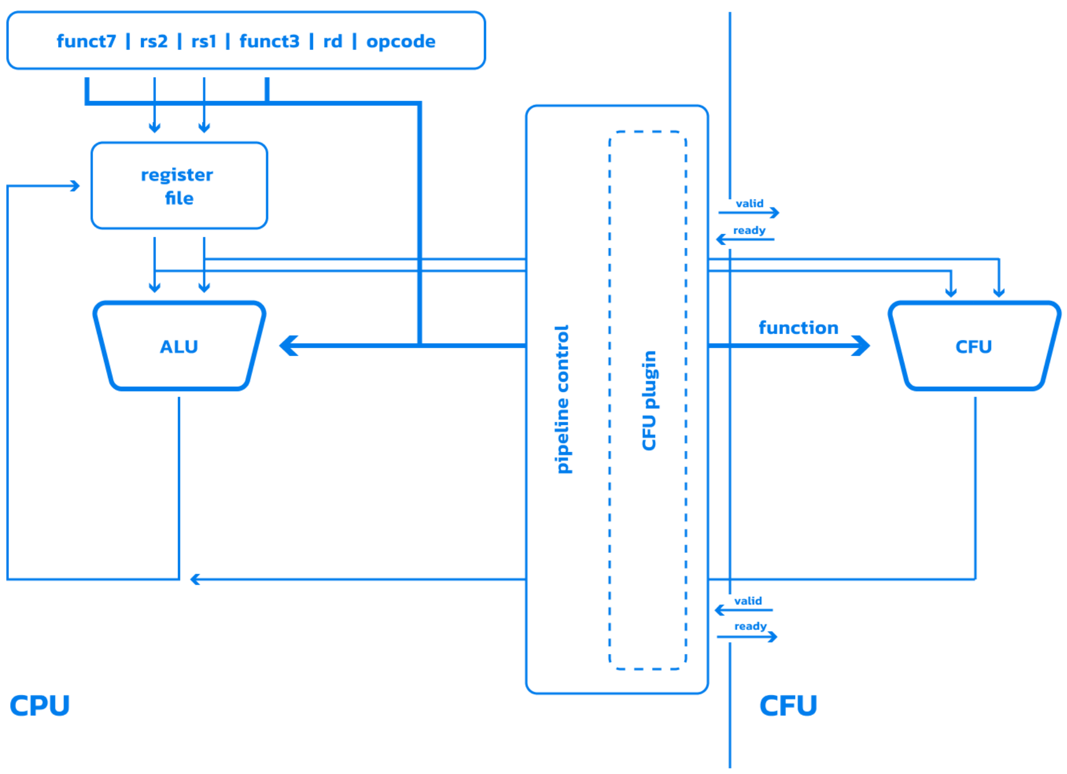


*Figure 6.2. CFU instruction flow diagram*

Regardless of the approach, Renode can leverage this data with its dts2repl tool [11], generating Renode platform description from Device Trees.

This work has become a basis for our other efforts, like massive Continuous Integration systems for Zephyr — the Zephyr Dashboard [20] or the U-Boot Dashboard [19]. The goal of these systems is to illustrate the breadth of hardware covered in Renode by using the framework to run automated test scenarios and display all supported targets for the given operating system or bootloader at any given moment. These systems also verify Zephyr and U-Boot themselves, and has since become a source of multiple improvements and bug reports to their upstream repositories.

## 6.3 CFU Simulation

The main aim of Task 5.3, as described earlier, was extending Renode's capabilities with support for RISC-V Custom Function Units (CFUs). Custom Function Units are AI accelerators for Field-Programmable Gate Arrays, tightly integrated with RISC-V CPUs via the custom instructions mechanism, exercising a custom instruction space reserved by the RISC-V ISA spec (Fig. 6.2). This specification provides a standard that covers complex operations common for such accelerators, encouraging reuse and cutting down time spent on defining and implementing this type of interfaces.

Support for CFUs is implemented in Renode via its co-simulation integration layer for Verilator, a free and open source Verilog/SystemVerilog simulator.

In a collaboration with Google, we have also implemented a Renode-based testing and verification flow for CFUs in an open source project named the CFU Playground [78]. CFU Playground is a framework designed for developing CFUs and reasoning about ML accelerators in simulation and using FPGAs. Thanks to Renode's modular nature, it is possible to automatically generate platform descriptions from CFU Playground samples, enabling automatic testing with every change to either the CFU, the SoC it works with or the software they run. The framework is based on VexRiscv [164], an open source RISC-V soft processor, optimized for FPGAs.

A CFU can be written in Verilog or any other language that outputs Verilog. In the CFU Playground demos, CFUs are mostly written in Amaranth [4], which allows you to write code in Python and generate Verilog output. The Python-based flow simplifies development for software engineers who may not be familiar with writing Verilog code. Since the code is generated from Python, it is also very easy to incrementally modify in a structured way until the expected acceleration targets are met.

## 6.4 Co-simulation Improvements

The improvements developed during this project are not just limited to CFUs, but rather extend to general co-simulation capabilities. These developments can be utilized for developing accelerators that are less tightly coupled with the CPU — accessible via bus accesses. AXI4 and AXI4Lite buses were our main focus within the project, however APB3 and Wishbone buses are supported as well.

In Deliverable D5.3 [202], we described our work around extending Renode with co-simulation capabilities with Verilator in order to support development of RISC-V-based ML accelerators. During this part of the project, we were able to improve the accuracy of transactions between the CPU and peripherals from the perspective of conformance with bus specification. We have also introduced developments in trace generation and model evaluation procedures, bridging the gap between the ways Verilator and Renode handle them, and making them easy to follow for hardware engineers. Another area where we introduced improvements was focused on scenarios where a peripheral converted via Verilator acts as both the initiator and as the recipient of a transaction. Here, by redesigning the simulation flow, we were able to significantly reduce the runtime for some of the test scenarios.

These improvements, however, were specific to Renode's integration with Verilator and use a bespoke interface. In order to be able to use Renode to co-simulate in tandem with most prevalent industry solutions, e.g. Questa, Vivado, we extended the framework with support for Direct Programming Interface (DPI) [10], a standard interface for connecting SystemVerilog models to native code - in our case used to interact with Renode.

While our custom Verilator interface works via sockets and native calls, the DPI interface is only available via sockets at the moment. The DPI interface supports AXI4, AXI4Lite, and APB3, with AHB underway.

Our work on this task in the recent period was focused on simplification and unification of examples provided with Renode [18] and improving the integration layer itself [17]. All changes are released publicly and are used by research and commercial partners.

## 6.5 Improvements to the Testing Infrastructure

Renode has been created with testing in mind, on all levels of complexity — from simple single core microcontrollers to large, multinode Linux-based systems with peripherals connected over wires and wirelessly.

The improvements to co-simulation integrations mentioned in the previous section allowed us to prepare two repositories containing examples that let developers easily recreate GitHub Continuous Integration flows and adjust them to their needs:

- renode-verilator-integration [16]
- renode-dpi-examples [15]

### 6.5.1 Renode-Verilator Integration

This repository contains samples of Verilog models and wrapper code that uses the co-simulation integration layer for Verilator that also implements support for CFUs. Developers can use this repository in order to generate their own models of peripherals, either accessible via bus or as CFU modules.

In order to create their own CFU unit, users need to fork the repository to their organization and copy one of the available sample.

If they prefer to create a sample from scratch, they must follow several unified steps described below. Please note, however, that the most important part of this period development focused on enabling users to simply copy an example and easily adjust it to their requirements.

First they need to make sure that the following C++ headers from the Renode Verilator Integration Library are included in the $main.cpp$ file for their model:

```
#include “src/renode_cfu.h”
#include “src/buses/cfu.h”
```

Then the $RenodeAgent$, the model's $top$ instance, and the $eval()$ function need to be initialized:

```
RenodeAgent *cfu;
Vcfu *top = new Vcfu;

void eval() {
   top->eval();
}
```

Optionally, users can include evaluation tracing code in the $eval()$ function, following the prepared samples.

Next, the $Init()$ function that initializes a bus and its signals needs to be added along the $eval()$ function. Upon initialization, it will return a $RenodeAgent$ object connected to a bus:

```
RenodeAgent *Init() {
    Cfu* bus = new Cfu();

    //=================================================
    // Init CFU signals
    //=================================================
    bus->req_valid = &top->cmd_valid;
    bus->req_ready = &top->cmd_ready;
    bus->req_func_id = (uint16_t *)&top->cmd_payload_function_id;
    bus->req_data0 = (uint32_t *)&top->cmd_payload_inputs_0;
    bus->req_data1 = (uint32_t *)&top->cmd_payload_inputs_1;
    bus->resp_valid = &top->rsp_valid;
    bus->resp_ready = &top->rsp_ready;
    bus->resp_ok = &top->rsp_payload_response_ok;
    bus->resp_data = (uint32_t *)&top->rsp_payload_outputs_0;
    bus->rst = &top->reset;
    bus->clk = &top->clk;

    //=================================================
    // Init eval function
    //=================================================
    bus->evaluateModel = &eval;

    //=================================================
    // Init peripheral
    //=================================================
    cfu = new RenodeAgent(bus);

    return cfu;
}
```

Before the project is compiled, three environment variables first need to be exported:

- $RENODE\_ROOT$ - path to Renode source directory
- $VERILATOR\_ROOT$ - path to the directory where Verilator is located (this is not needed if Verilator is installed system-wide)
- $SRC\_PATH$ - path to the directory containing your $main.cpp$

With these variables in place, it is now possible to build the CFU in $SRC_PATH$:

```
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release "$SRC_PATH"
make libVtop
```

In order to attach the verilated CFU to a Renode platform, the $CFUVerilatedPeripheral$ needs to be added to the $RISC-V$ CPU as follows:

```
cpu: CPU.VexRiscv @ sysbus
    cpuType: "rv32im"

cfu0: Verilated.CFUVerilatedPeripheral @ cpu 0
    frequency: 100000000
```

Finally, a path to the compiled CFU is required. It can be provided in the $.repl$ platform or using a $.resc$ script.

### 6.5.2 Renode DPI Examples

This repository contains samples that leverage the integration between Renode and a Verilog model via DPI calls. All samples in the repository are designed to work with Verilator as well as Questa. The CI flow included in the repository is configured to build and run samples in Verilator and the CI configuration can serve as a point of departure and a reference for preparing other examples. The repository contains samples for peripherals connected over AXI4, AXI4Lite, and APB3.

Once a user has Verilator built and Renode installed, they can build one of the sample peripherals and confirm that the communication of the Verilated peripheral with Renode behaves as expected by using one of the Robot Framework tests present in the repository, which connects to either Verilator or Questa and performs operations specific to the simulated block.

Developing a completely open-source workflow allowed us to easily integrate the new features into a CI pipeline that automatically builds and tests the DPI integration against samples of varying complexity on different target OSes, for example Linux on Zynq-7000 with a verilated FastVDMA controller. The $renode-dpi-examples$ repository lets you run such a demo using a single com mand:

```
renode/renode samples/axi_fastvdma_prebuilt/platform.resc
```

This lets developers verify their Verilog models upon every single change, so that they can easily find, recreate, and fix problems in a deterministic, controlled environment.

As with Verilator-based exmples, DPI repository encourages user to fork and adjust one of the samples to their needs.

The SystemVerilog-based integration is slightly more involved when creating a sample from scratch.

With the code ready the compilation process is unified for all samples:

```
cd {path_to_a_sample}
mkdir build
cd build
cmake .. -DUSER_RENODE_DIR={path_to_renode_instrallation} \
     -DUSER_VERILATOR_DIR={path_to_verilator_installation}
make
```

Please note that due to significant improvements introduced to Verilator itself during the course of the project (mostly related to issues handling SystemVerilog code), Verilator version v5.010 is recommended.

## 6.6 Coherent Co-simulation Flow for ML Development

Work performed during the VEDLIoT project allowed us to greatly increase Renode capabilities in the co-simulation space.

Not only were we able to improve support for two types of co-simulation interfaces, to introduce support for CFU Machine Learning accelerators and to prepare example repositories with multiple samples and tests, we also verified the usefulness and correctness of our implementations in the commercial context.

Multiple blog posts [9] described created developments and gathered significant interest, also leading to external contributions towards Renode in the area of co-simulation capabilities.

In future co-simulation will remain one of the most important use-cases for the Renode Framework, with expected improvements to ease of use, range of supported scenarios and performance of simulation.

# 7 Robustness for Deep Neural Networks

During this last period of the VEDLIoT project, as part of task 5.5, our focus was on implementing two algorithms aimed at verifying and quantifying, as well as enhancing the resilience of deep neural networks (DNNs) against adversarial attacks. These algorithms seek to ensure the robustness of DNNs without compromising their predictive accuracy. They were designed and could be used in the Automatic Emergency Braking With Pedestrian Detection (AEB-ped) use case within the project, where reliability is critical. Another use case where these algorithms can be used is related to the smart mirror, allowing improved gesture recognition. In a generic application of these algorithms, they can be integrated with any other use case involving DNNs.

The designed algorithms were aimed to withstand potential threats and serve as critical components in safeguarding the integrity and effectiveness of AEB-ped systems, reflecting our commitment to advancing the robustness and reliability of deep-learning technology. Our contributions align seamlessly with the broader VEDLIoT mission. These algorithms can be used to address the need for robust, secure, and efficient deep-learning technologies in autonomous vehicles, explicitly emphasizing the critical aspect of automatic emergency braking with pedestrian detection.

Complementary to these algorithms, we are implementing a self-ensemble horizontal gating-based mixture of experts image classification deep learning model. The idea is to implement a framework able to handle diverse driving scenarios, considering adversarial and drift inputs. This is ongoing work. However, in this deliverable, we describe the framework and its mechanisms.

## 7.1 Robustness Verification of Neural Networks

Deep neural networks (DNNs) are characterized by their nonlinearity and large-scale architecture, making it challenging to analyze their behaviors comprehensively as to why a particular decision has been made. They are often deployed as ”black box” models without formal robustness and safety assurances. Therefore, verifying their robustness to potential real-world scenarios is pivotal and necessary for their development in safety-critical systems such as self-driving cars. In particular, NNs are not robust to input perturbations (adversarial examples), which makes them highly vulnerable to adversarial attacks in run-time settings. For example, in the context of image classification in a self-driving car, the neural vision of a well-trained NN can be easily fooled if an imperceptible perturbation is applied to the same image it was trained on without altering the entire representation of the image. In such scenarios, the scene analysis layer could confidently analyze a stop traffic sign as a speed limit traffic sign with high confidence in the driving environment. As a result, the decision-making model could generate misleading steering commands, which could lead to catastrophic consequences [131]. Thus, it is essential to formally understand the areas where NNs may be likely to wane in their decisions under extreme scenarios and estimate their sensitivity level under variations in the input space.

As part of this report, we investigate and design a network block segmentation approach that divides the entire hidden layer of the network into segments parallel to each other, applies the $\alpha$-$\beta$-CROWN [33] algorithm to each segment, maps together the outputs of all segments, and computes the overall verification score at the final output layer. Our approach improves the scalability of the $\alpha$-$\beta$-CROWN verification tool, reduces the need for high computational demands, and speeds up the verification process on large-scale NNs in a Branch and Bound (BaB) [95] setting. Interestingly, compared to applying the standard $\alpha$-$\beta$-CROWN verifier to the entire network simultaneously, our proposed approach delivers superior verification scores and generates tighter certified bounds for ReLU-neurons in each block. It requires less computational time to generate the quantification score of the model's robustness level to input perturbation.

### 7.1.1 Related Works

Three dimensions may be used to evaluate the effectiveness of robustness verification algorithms for NNs. The first dimension measures the tightness of the verification constraints (lower and upper bounds), the second measures the computational complexity, and the third measures how compatible it is with DL models using different state-of-the-art activation functions, such as ReLU, Tanh, Leaky ReLU, Sigmoid, Parameterized ReLU, and ELU. Lomuscio et al. [116] leveraged mixed-integer programming (MIP) algorithms to perform exact robustness verification of a nonlinear DNN, which they claimed reduced the computational burden and complexity of verifying the robustness of NNs. However, their solution is not optimal for generating tighter linear bounds and is only constrained to nonlinear NNs. In a similar work, Kwiatkowska et al. [106] suggested an automated DNN safety and robustness verification approach. The author's methodology is based on Bayesian techniques, feature-guided search, and global optimization to verify the robustness of NNs. Wong et al. [193] introduced a linear programming (LP) relaxation of piece-wise linear NN and set upper bounds on the worst-case loss by employing less strong duality to verify the robustness of the network. Similarly, Raghunathan et al. [153] proposed a semidefinite programming (SDP) convex relaxation of a single-layered sigmoid-based NN that employs a first-order Taylor expansion to bound the worst-case losses. Their method is not computationally efficient for verifying the robustness of large-scale DNNs.

To our knowledge, the only convincing algorithm that tries to address these limitations by solving the split constraint imposed by the BaB algorithm and verifying the robustness of large-scale NN is $\alpha$-$\beta$-CROWN [33] verifier. This verifier received the highest verification score in the $2021$ and $2022$ International Verification of Neural Networks Competition. However, this study shows that $\alpha$-$\beta$-CROWN is not as scalable as it appears on large-scale NNs because it requires a lot of computational power to compute the Lipschitz constant of the network's output concerning the input. To address this limitation, we have a scalable approach that produces tighter linear bounds with fewer computational resources compared to using the standard $\alpha$-$\beta$-CROWN over the entire network.

### 7.1.2 Robustness Verification and Quantification Approach

The traditional CROWN verifier was the most commonly used incomplete verification tool for complementing the complete verification process in adversarial settings. However, it is too weak to satisfy the split constraint imposed by the BaB method because CROWN operates entirely on bound propagation, similar to the backpropagation [194] in the normal NN. However, it is pretty effective in the incomplete verification process but too weak in the complete verification process. In this context, incomplete verification algorithms are verifiers that can indicate whether an NN is robust but cannot offer or indicate the robustness level of the network. In contrast, complete verification algorithms can indicate or prove the robustness state of an NN and, at the same time, quantify the robustness level of the network.

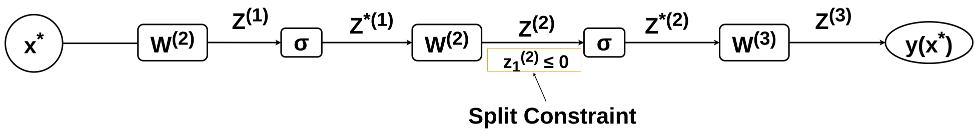


*Figure 7.1. Block diagram of a DNN*

Using CROWN, if the requirement that $y^*_{CROWN} > 0 \implies y_{min} > 0$ is satisfied, the network's robustness can be guaranteed for all inputs. In a backward bound propagation, CROWN computes the network's output $y(x^*)$ by linearizing the input of the ReLU activation to determine the lower bound $(y_{min})$ of each ReLU-neuron. This entails relaxing the convexity so that the network's input and output have a linear relationship. Hence, in a CROWN setting, $y(x^*)$ is express as:

$$y(x^*) = W^3\sigma\left(W^2\sigma\left(W^1x^*\right)\right) \tag{7.1}$$

Where;

$W$ is the weight of the network

$y(x^*)$ is the output of the network concerning the perturbed input $\sigma$ is the non-linear activation function

Given that, we can now compute the CROWN-based linear lower bound of the final output neurons as:

$$y^*_{CROWN} = -||a_{CROWN}||_\infty + a^T_{CROWN}x^* + Const_{CROWN} \tag{7.2}$$

The $y^*_{CROWN}$ represents the verification score value obtained by the CROWN algorithm considering $x^*$. The $-||a_{CROWN}||_\infty$ represents the negative product of the $L_\infty$ norm of $a_{CROWN}$. The constant $\epsilon$ signifies a penalty term based on the $L_\infty$ norm or a factor that controls the magnitude of the $L_\infty$ norm. $a^T_{CROWN}x^*$ represents the dot product of the transpose of the vector $a_{CROWN}$ and perturbed input $x^*$. Finally, the $Const_{CROWN}$ represents a small constant value, which allows the network to capture complex relationships between $x^*$ and $y^*_{CROWN}$ more effectively.

Although we have proven that CROWN established a linear relationship between the output and the input of the network as it computes the lower bounds of the final output neurons in a backward bound propagation as shown in equation (7.2). However, it cannot optimize the neuron-split constraint introduced by BaB because it generates certain slack bounds that prevent the convex from being relaxed appropriately. Therefore, it could not provide an accurate robustness guarantee to the model for all inputs.

Therefore, using our benchmark datasets, we experiment with the $\alpha$-CROWN incomplete verifier proposed by Xu et al. [197], an extension of the traditional CROWN verifier that takes advantage of reverse-phase linear relaxation-based perturbation analysis (LiRPA) during the BaB process.

We consider the CROWN-based linear lower bound in equation (7.2) as a function of $\alpha$because since $\alpha$is an adjustable parameter, it can generate different lower bounds, among which we can use any of them as long as its slope exists between $0$ and $1$ across the origin. Our experiment shows that $\alpha$-CROWN can generate tighter linear lower bounds than the CROWN algorithm.

The linear lower bound of a ReLU-neuron in an $\alpha$-CROWN setting can be expressed as;

$$y^* = \underset{x^* \in C}{Min}\, y(x^*) \geq \boxed{\underset{0 \leq \alpha \leq 1}{Max}}\, \underset{x^* \in C}{Min}\, y_{CROWN}(x; \alpha) \tag{7.3}$$

Although $\alpha$-CROWN is a good verification tool, generating a verification score for a single image often takes several iterations. It also requires higher computational time to provide an accurate robustness verification score in a BaB setting due to the split constraint when $z_1 \leq 0$ during backward bound propagation.

Due to these limitations of the $\alpha$-CROWN verifier, we further leverage the $\beta$-CROWN verifier proposed by Wang et al. [188], an advanced incomplete NN verifier, which is also an extension of the traditional CROWN verifier but quicker and more flexible compared to the $\alpha$-CROWN algorithm for solving the split constraints (sub-problems) in a BaB setting when $z_1 \leq 0$.

It encapsulates neuron split constraints in a BaB setting via an optimizable parameter ($\beta$), which is a "Lagrangian multiplier" [32] built from either primeval or dual dimensions. The "Lagrangian multiplier" always conditioned the split constraint $(z_1 > 0)$ to be either $0$ or $1$, making $\beta$-CROWN faster and producing better tighter bounds compared to the $\alpha$-CROWN algorithm while remaining as efficient and parallelizable as CROWN on GPUs [188].

The mathematical expression for $\beta$-CROWN is given as:

$$\begin{aligned}\underset{x \in C,\, z^{(2)} < 0}{Min}\, y(x^*) \geq \underset{\beta \geq 0}{Max}\underset{x \in C}{Min}\, w^{(3)T}\, D^{(2)}\, z^{(2)} + \\ + \boxed{\beta^T\, S^{(2)}\, z^{(2)}} + Const.\end{aligned} \tag{7.4}$$

Where $\boxed{\beta^T\, S^{(2)}\, z^{(2)}}$ is the Lagrangian multipliers in which $S$ is a diagonal matrix with $+/-1$ and 0.

According to our experiment, besides the $\beta$-CROWN verifier, the $\alpha$-$\beta$-CROWN [33] is the fastest and most accurate NN verifier that works well on powerful GPUs to provide robustness verification of large-scale NNs in a BaB setting. However, it requires a lot of GPU power to be scalable on large-scale networks. It combines the $\alpha$and the $\beta$parameters in a traditional CROWN setting. Its application in a BaB setting produces a complete verification algorithm, which indicates the network's robustness status and its level of robustness at the same time. $\alpha$-$\beta$-CROWN is expressed as:

$$\underset{x\in C,\ z^{(2)}<0}{Min}\ y(x^*) \geq \underset{\alpha\geq 0,\ \beta\geq 0}{Max}\ \underset{x\in C}{Min}\, w^{(3)T}\, D^{(2)}\, z^{(2)} + \\ + \boxed{\alpha^T,\ \beta^T\, S^{(2)}\, z^{(2)}} + Const. \tag{7.5}$$

Where $\boxed{\alpha^T,\ \beta^T\, S^{(2)}\, z^{(2)}}$ is the Lagrangian multipliers in which $S$ is a diagonal matrix with $+/-1$ and 0.

We developed a network block segmentation approach to divide the network into three parallel segments. Instead of applying the $\alpha$-$\beta$-CROWN algorithm to the entire network in each forward or backward pass, we applied it to each segment and mapped the outputs. This technique improved the $\alpha$-$\beta$-CROWN's scalability and minimized the requirement for the heavy computational demands imposed by its standard use, as proposed in [33].

## 7.2 Mitigation of Adversarial Data Poison Attacks Against Deep Neural Networks

In terms of mitigation of adversarial poison attacks in deep neural networks, we implement a novel method of robustifying a distilled network against data poisoning attacks by integrating a denoising autoencoder (DAE) [34] in the defensive distillation pipeline. Defensive distillation involves training two DNNs, the instructor model and the student model, such that the knowledge of the former is transferred into the latter, making it robust to adversarial examples and previously unseen input [86, 46]. A DAE is a type of DNN trained in an unsupervised setting to learn a latent representation of input data $x$, enabling it to reconstruct distorted inputs back to their original form. It learns how to reconstruct the perturbed version of $x$ while removing noise and reducing the reconstruction loss between $x$ and the reconstructed input $x_r$ as much as possible during the training phase [34].

The necessity of DNNs' robustness against adversarial attacks has shown a growing concern as their use in real-world safety-critical systems has increased exponentially. Adversarial attacks [46] attempt to trick DNNs by making subtle alterations to the input sample $x$. Several defence strategies have been proposed to overcome this problem, including defensive distillation, which has successfully defended DNNs from input perturbations $\varepsilon$ in run-time settings [143]. Nevertheless, one of the drawbacks of the defensive distillation technique is that it remains susceptible to data poisoning attacks, in which adversaries aim to impair the model's performance by either maliciously altering the existing training data or inserting erroneous data entries into it.

We used the fast gradient sign method (FGSM) [77] and the iterative-fast gradient sign method (I-FGSM) [104] to evaluate the effectiveness of the proposed algorithm.

The proposed approach offers a more potent protection mechanism against data poisoning attacks. It enables the distillation algorithm to mask or lower the gradient around the input space and widen the search space that attackers need to explore to craft adversarial examples in the input space $X$. Our approach significantly reduces the limitations and susceptibility of the defensive distillation algorithm to data poisoning attacks.

### 7.2.1 Related Works

Knowledge distillation was first proposed in [86]. The author aimed to lower the computational resources required to deploy large-scale DNNs on resource-constrained devices such as smartphones. Therefore, they extrapolate the probability vector or class knowledge produced by the instructor network and use it to train small networks, reducing the network's scalability without compromising accuracy and allowing deployment on resource-constrained devices. Statistically and experimentally, Papernot et al. [143] further explore this idea as a preventive measure against adversarial inputs. Using class knowledge from the instructor network and distilling it to the student network, the authors minimize the amplitude of the student network's gradients that attackers required access to generate adversarial examples in the input space $X$. A similar approach was Goldblum et al., [74], they developed an Adversarially Robust Distillation (ARD), which involves creating small NNs and distilling their robustness in a more extensive network. Previous works leveraging the defensive distillation technique include [105, 44, 142]. Yue et al. [199] demonstrated that DAEs are highly helpful in spotting and reconstructing contaminated images in the training data manifold. They apply this approach to identify and mitigate adversarial poison attacks in the federated learning setting. Other works in which DAE is used as a filter against poison data include [98, 174].

### 7.2.2 Mitigation of Adversarial Data Poison Attacks

Defensive distillation involves training two homogeneous or non-homogeneous DNNs: the instructor network $F\phi_1$ and the student network $F\phi_2$. This work uses a standard NN training procedure to train $F\phi_1$ using the original dataset, increasing the softmax temperature $T$ to $5$ degrees.

The softmax layer normalizes an output vector or logits $Z(X)$ of the $F\phi_1$ final hidden layer into a probability vector $F(X)$ more closely aligned with the data manifold's uniform statistical distribution. This assigns a probability value to each class of the dataset for each input $x$. A specific neuron in the softmax layer that corresponds to a class indexed by $i0....N–1$, where $(N = numberofclasses)$ calculates element $i$ of the probability vector given by;

$$F(X) = \left[ \frac{e^{Z_i \frac{(X)}{T}}}{\sum_{l=0}^{N-1} e^{Z_l \frac{(X)}{T}}} \right]_{i \in 0----N-1} \tag{7.6}$$

Where;

$X$ is the input space

$F(X)$ is probability vector

$T$ is the softmax temperature parameter

The $F(X)$ is used as a label for the inputs in the training data to train $F\phi_2$ with the same temperature value as used in $F\phi_2$. $T$ is usually reset to its default value of $1$ during testing so that the distilled network can produce more discrete output. This also discourages overconfidence in the distilled network's output and improves its generalization to new inputs.

A Denoising Autoencoder (DAE) is designed to learn a compressed representation of $x$, correct any abnormalities in the data, and reconstruct it back to its original, undistorted state with the help of the latent vector $h$. The design comprises an encoder network $f_e$ and a decoder network $f_d$, represented as a composition of $i-th$ encoding layer $f_e^{(i)}$ and decoding layer $f_d^{(i)}$, respectively. During training, the first layer of $f_e$ receives the erroneous form of data $x^*$ from the input space $X$ and translates it to a lower-dimensional latent vector $h^{(i)}$. The subsequent layer maps $h^{(i)}$ from the previous layer to generate more compressed latent features. This process continues until the last layer, which represents the final lower-dimensional latent representation of the input $h$. Conversely, the last layer of $f_d$ takes in $h$. It maps it to the next layer in a backward pass direction until the first layer, which maps the reconstructed features to the original input space, producing the reconstructed output image $x_r$. The latent representation of the $i-th$ layer can be expressed as;

$$h^{(i)} = f_e^{(i)}(h^{(i-1)}), \quad where h^{(0)} = x \tag{7.7}$$

where;

$h^{(i)} =$ latent representation at the $i-th$ encoder layer.
$i =$ the layer index of the encoder
$f_e^(i) =$ composition of individual encoder functions

Adversarial attacks are types of malicious data modification that seek to deceive ML models in their decision-making [46]. In contrast, adversarial robustness describes the model's capacity to maintain its expected performance in the presence of malicious interruptions or adversarial attacks [56]. A model is deemed robust if it correctly classifies $x \in X$ within or outside a range of perturbation sets defined in $X$. In our experiment, we used the following parameters to generate adversarial perturbations for FGSM and IFGSM; $\varepsilon_{fgsm} = 0.01, \varepsilon_{ifgsm} = 0.01, \alpha = 0.01, num\_iterations = 10$. The epsilon $\varepsilon$ determines the magnitude of the perturbation needed to trigger the model's sensitivity to changes in the image's pixel values. The alpha $\alpha$parameter controls the step size of the perturbations in each iteration of the IFGSM attack. $num_iterations = 10$ indicates that the IFGSM will be applied at every $10$ iteration.

Following the methodology used in [142], the student model in our experiment is built to be uncertain about its prediction when a new input $x_n$ is statistically different from the training set (i.e., discouraging overconfidence in its classification). We used the Kullback–Leibler (KL) divergence [154] to quantify the statistical differences between the $x_n$ and the ground truth. This approach allowed the model to offer uncertainty estimates for the predictions and estimate Bayesian inference. Analyzing the instructor model's predictions yields the uncertainty measurements needed to train the student model.

The effectiveness of the trained DAE is evaluated on a different test dataset containing both clean and distorted images. A specified reconstruction threshold is set, which serves as a yardstick or decision line to verify the integrity of each data point after reconstruction. The images whose reconstruction error is above the threshold are termed adversarial inputs and are subsequently discarded before reaching the teacher model.

## 7.3 Self-ensemble Horizontal Gating-based Mixture of Expert Image Classification Deep Neural Networks

Complementary to the work done in Sections 7.1 and 7.2, we are implementing a self-ensemble horizontal gating-based mixture of expert image classification deep learning models that should be able to handle diverse driving scenarios and be able to handle adversarial and drift input in adverse situations. A mixture of experts (MOE) model is a machine learning technique that uses different neural networks, each specializing in a different part of the training, and a gate network that is trained on the entire input space, enabling it to assign different weights to each different expert based on their contribution during inference. We are currently in the preliminary stage of this work, where our work is currently limited to the use of the German Traffic Sign Recognition Benchmark (GTSRB) dataset. We exclusively train and evaluate 43 expert models and one gate network, where each expert is trained only on a particular class within the GTSRB dataset.

We introduced a dynamic batch size and learning rate technique where different batch sizes and learning rates are assigned to experts based on the number of samples they receive as inputs in the class folder. The rationale behind this technique is that we want to make sure an appropriate batch size and learning rate are assigned to each expert based on the number of training samples in its corresponding class folder rather than using a fixed batch size and learning rate for all the experts, which may introduce some bias into some experts, especially those with lesser number of samples since we are currently dealing with an unbalanced dataset with a high proportion.

Assigning each expert to a specific class within the dataset allows it to specialize in learning and understanding the features relevant to that class. However, with a fixed batch size, the number of samples from each class per iteration is the same. This means that experts in classes with fewer samples might not receive sufficient data to specialize effectively. If a fixed batch size is used, the classes not presented with enough examples during training may not be well generalized by the overall model. This can lead to poor performance in infrequent classes during inference. Therefore, experimenting with different setups and monitoring the performance of each expert in their associated classes can help find an effective configuration for the MoE model when dealing with large datasets.

We currently use a simple LeNet5 deep learning model for all the experts. In the second phase of this work, we will use a complex dataset that covers all the inputs from different driving scenarios and conditions. Each expert may be trained on multiple classes since the dataset will be clustered based on input-output mapping. In that stage, we will improve the scalability of the experts by using large-scale networks such as the ResNet family (ResNet 50, 100, 120, etc.) or the VGG family (VGG 19, 16, 18, 32, 64, 101, etc.).

The third stage of this work will involve evaluating the robustness of the MOE framework by verifying and quantifying its robustness by introducing different types of adversarial attacks and data drift in the evaluation or testing phase. We will determine the robustness of the framework by evaluating its ability to generalize on unseen inputs in the presence of adversarial and drift inputs in the run-time setting.

# 8 Generating, Sharing and Using Data for Increased Safety – Advantages and Challenges

In this chapter we discuss the possibility of sharing data between vehicles in a traffic scenario to increase road safety. The availability of data received from surrounding vehicles can be used to improve perception, namely by fusing it with data collected from local sensors. For instance, data about pedestrian detection, which is produced in the Automatic Emergency Braking With Pedestrian Detection use case, could be exchanged with other vehicles in the vicinity, hence contributing to increasing the accuracy of pedestrian detection, and possibly serving as input for other functions. However, exchanging data with other vehicles also raises some problems, namely of data trustworthiness. Therefore, in the following sections we discuss the several facets of this problem, laying down the path for possible future work.

## 8.1 Reasons for Sharing Data

There are many potential benefits of sharing data in the traffic environment and using it to improve safety and flow. First comes reducing the number of accidents and the severity of those that do occur. Considering motorized vehicles, the ride could be made more comfortable with smoother accelerations and less harsh braking which in turn could reduce the energy consumption and the emissions. The environmental impact can even be said to go as far as making the traffic environment more quiet. By improving the traffic flow, the time spent on transportation could be reduced and make room for other activities in people's lives. The benefits of the concept, which is called many things e.g., Collective Perception, Collaborative Driving, Collective Safety to mention a few, are obvious, but the implementation has proven to be difficult.

There are several dimensions of challenges that need to be addressed. A few of them are that the vehicle fleet is heterogeneous and there are no all-encompassing regulations concerning the generation and sharing of information. Also, the traffic environment includes vulnerable road users, e.g., pedestrians and cyclists, that don't benefit from the same type of decision support that a vehicle can provide its driver. Furthermore, suitable infrastructure has to be deployed and its use by different parties negotiated. In short, aspects concerning business, engineering, safety and legal matters have to be handled before data sharing on a big scale can become reality.

## 8.2 Available Sharing Options/Infrastructure

The idea of sharing data to avoid incidents and accidents is not new and there are fully operation systems for distribution of information and work is ongoing to introduce more recently developed and more technologically advanced systems.

### 8.2.1 Broadcast

There are systems in use (Radio Data System in Europe, Radio Broadcast Data System in North America) where signals are added to broadcast FM radio to transmit messages to relevant to the near geographical area. Road uses can report incidents which is then distributed, but this is mostly a one-way communication that usually reaches more users than necessary.

### 8.2.2 Two-way Communication

Currently, state of the art in wireless two-way communications for traffic purposes is developing in two branches. First, dedicated short-range communications (DSRC) for Vehicle to Everything (V2X) is a wireless two-way communications technology, operating at 5.9 GHz and based on IEEE 802.11p, and second, cellular V2X, C-V2X which uses cellular communications, where 3GPP are developing the standards. Both branches offer the ability to establish communications directly between two vehicles, but C-V2X also has the option to communicate with the network V2N and thus with non-vehicle user equipment, e.g., cellphones.

Work is ongoing within the Hexa-X-II project [180] where communication and sensing is one field that is being studied in a 6G-context.

## 8.3 Creating the Big Picture

One intermediate goal that supports traffic safety is to generate what can be called a ”Recognized Ground Picture” which contains information on all entities in a traffic situation. This RGP would be the result of a combination Then the question arises: ”What information is relevant for entity number n?” In order for information to be truly useful it must be correct, timely and relevant. Only then can it be most beneficial to traffic participants. In this section the perspective starts from a single vehicle, expanding to encompass an entire traffic situation.

### 8.3.1 Generating Data and Local Information

Data about the traffic situation can be generated in several ways. Vehicles are becoming equipped with increasingly advanced sensor systems using cameras, radar and even infra-red imaging and lidar in some cases which makes it possible to simultaneously use data from different sensor modalities and obtain information that can be used to support the driver’s situational awareness. The downside is that the reach of the vehicle’s sensors is limited which makes the benefits of including external data clear as it can provide information about obscured dangers. External data could be generated by other vehicles, but sensors can also be found in traffic infrastructures, e.g., cameras for monitoring road conditions and radars for velocity that are also becoming prolific in some countries. Communication devices, e.g., cellular phones, could also provide additional data.

When generating data, it is important to consider privacy, both for personal and legal reasons. Traffic participants must be able to trust the system to only use the data to increase the safety in the local traffic system for the time it is needed.

### 8.3.2 Sharing and Processing

Turning data from single sources into something bigger and useful requires data transfer. When data becomes information is subject for discussion and no clear definition is commonly accepted. Each sensor generates data that can undergo further processing and at some level, be fused with other data. How to do this in the best way is a field in its own right, and a challenge even if the data is generated by sensors attached to one single vehicle. Fusing data/information from several different vehicles is even more difficult because there is a need to relate the entities and the data they provide to each other in both space and time and also to geographical information. This puts requirements on the sensing equipment, the ability to transfer and process data.

First, the data must be associated with some kind of quality measure to ensure that the accuracy and robustness of the resulting information is upheld. As the vehicle fleet is heterogeneous and there traffic participants of all kinds, from vulnerable road users e.g., pedestrians, bicyclists, to large transport trucks, the data can for example have different levels of resolution, accuracy and update rate.

Second, the data transfer must have a sufficiently high capacity and be organized, or orchestrated, so that the resources are distributed to maximize throughput. It is important to consider what data to share and how in order have a reasonable margin left in the systems and also to consider the energy consumption for the service.

Third, the processing of the data requires computational power which can be found both onboard vehicles but even more importantly, in communications infrastructure. The latter may offer computational power as a service which could provide an increased equality in the traffic environment as vehicles with less advanced processing equipment would be able to contribute and benefit.

The next step is the continuously ongoing process is to distribute relevant data to road users so that they can either incorporate the information into their sensor data processing or act upon it.

### 8.3.3 Distributing and Using Information

Assuming that trustworthy information has been generated in time to be useful, it needs to be distributed to the road users. It can be done locally broadcasting information which is tagged to be relevant for a limited geographical area, or even tailored specifically to particular vehicles, depending on how the distribution is implemented.

Depending on the needs of an individual traffic participant, the information can be used directly, for example an e-scooter driver receiving a warning about crossing traffic. In this case, the own vehicle may not contain any relevant sensors, but could have means of communication to receive information. If the consumer of the information uses a vehicle with a sensor system it may be beneficial to combine the received information with current vehicle sensor information or in some instances, perform a peer-to-peer information sharing. The latter is important in traffic environments where the communications networks are less dense, which is often the case in rural areas, or if the network for some reason is not operational.

When consuming information, being able to trust its quality and reliability is of utmost importance. When the vehicle's own sensor systems are used, the functional safety aspects concern a system with known characteristics. Incorporating data from other sources can have implications on safety as well as legal matters, which can be one of the reasons that information sharing in the traffic environment is still limited.

## 8.4 Outlook

To obtain the advantages outlined in Section 8.1 there is a need for an infrastructure than can provide secure communication, computational power and the ability to orchestrate the data flow. As nothing comes for free, the question is what the business case looks like. Using cellular communications infrastructure is most likely the most cost-effective way that would provide the best coverage. Emergency services work across the boundaries of different network operators, which could be seen as example of a successful implementation.

In order for a system to be efficient and actually be used it needs to provide a clear advantage to the users, which means encompassing several ways of using it. Granted, all traffic participants will not get the full advantage of the system, either because their vehicle is not fully equipped or because the information cannot be shared to a human as fast as it can a car with advanced driver-assistance systems and automated driving. A realistic goal can be to share as much as data as is deemed possible and by preventing vehicles from colliding with vulnerable road users, much can be gained. In a long term perspective, information sharing services are likely to be a key component both to support safe automated driving.

# 9 Overall Achievements

In this deliverable we presented the work done in WP5 during the last part of the project. This included the presentation of extended evaluation results (e.g., SIRE evaluation), improvements and additions to previously presented tools (e.g., Renode improvements), discussion of paths ahead (e.g., use of shared data in autonomous driving scenarios), new and improved methods (e.g., for ML robustness), and completion of several solutions for computation and communication security (e.g., Twine and Fortress).

In a very objective and systematized way, some of the key contributions of WP5 to advance the state of the art on security, safety and robustness in IoT applications are listed and briefly described below. A complete and more detailed list of achievements is provided in Deliverable D1.4.

The contributions listed consist in several building blocks that can be used in ML-based IoT applications, and some were developed and or demonstrated in the context of the defined VEDLIoT use cases, namely in the context of the Automatic Emergency Braking With Pedestrian Detection use case, the Motor Condition Classification use case, and the Arc Detection use case.

The key contributions from Task 5.1 were the following:

- **Extended the TLS handshake** in webAssembly runtime, to include attestation.
- **SIRE**, providing distributed Byzantine Fault-Tolerant attestation and membership management.
- **AutoCert**, providing TOCTOU-security by combining Remote Attestation results about assurance of device health with standard Public Key Infrastructure (PKI) authentication processes.

The main building blocks produced by Task 5.2 were the following:

- **Twine (and Twine2)**, which is an execution environment suited for running Wasm applications inside TEEs.
- **WaTZ**, which is an efficient and secure runtime for trusted execution of Wasm code inside TrustZone, adding support for remote attestation.
- **Identification of vulnerabilities in TrustZone-M** and proposal of efficient solutions.
- **Fortress**, which is a robust and comprehensive framework to enhance security and privacy in IoT infrastructures.
- **Secure publish/subscribe** solution applied to the Mosquitto MQTT broker.
- **Improved security of the "Secure IoT Gateway"** VPN management solution.

The development of a simulation platform in the context of Task 5.3 and the continuous integration workflow aimed by Task 5.4 were addressed closely together by means of the following contributions:

- **Renode Custom Function Unit** support
- **Renode co-simulation framework** improvements

Finally, Task 5.5 produced the following main contributions:

- **Definition of safety methods for AI/ML systems**, including safety requirements, safety verification and safety runtime methods.
- **Definition of strategies for monitoring and mitigation of run-time errors** in the context of pedestrian detection in autonomous driving systems, for increased reliability and safety.
- **Methods for increasing the robustness of image processing DNNs**, in the context of autonomous driving systems.

Overall, given: a) the successful definition and development of several solution contributing to advance the state of the art, substantiated not only by WP5 deliverables, but mainly by the many achieved publications (as listed in Deliverable D8.4); b) the integration and demonstrating of several of the developed building blocks in the VEDLIoT use cases; and c) the fact that the KPIs defined for WP5 were met (as presented in Deliverable D1.4), we conclude that the work done in WP5 was of high quality and the main objectives set forth for the work package were fully achieved.

# 10 References

[1] MalwareTextDB: A Database for Annotated Malware Articles. https://www.aclweb.org/anthology/P17-1143/, July 2017.

[2] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. OBLIVIATE: A data oblivious filesystem for Intel SGX. In *NDSS '18*, NDSS '18. The Internet Society, 2018.

[3] A. K. M. Mubashwir Alam and Keke Chen. Making your program oblivious: a comparative study for side-channel-safe confidential computing. *CoRR*, abs/2308.06442, 2023.

[4] Amaranth community. Amaranth hdl. `https://github.com/amaranth-lang/amaranth`. Accessed on: Jan. 29, 2024.

[5] Amazon. X.509 client certificates, Mar 2021.

[6] Amazon. Pub/sub messaging, 2023.

[7] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. *White Paper*, page 20, January 2020.

[8] Tim Ansell, Tim Callahan, Jan Gray, Karol Gugala, Maciej Kurc, Guy Lemieux, Charles Papon, Zdenek Prikryl, and Tim Vogt. Draft proposed risc-v composable custom extensions specification. `https://github.com/grayresearch/CFU/blob/main/spec/spec.pdf`.

[9] Antmicro. Antmicro blog. `https://antmicro.com/blog/`. Accessed on: Jan. 3, 2024.

[10] Antmicro. Dpi support in renode for hdl co-simulation with verilator and questa. `https://u-boot-dashboard.renode.io`. Accessed on: Jan. 4, 2024.

[11] Antmicro. Dts2repl. `https://github.com/antmicro/dts2repl`. Accessed on: Jan. 3, 2024.

[12] Antmicro. Enabling secure open source ml products with open se cura. `https://antmicro.com/blog/2023/11/secure-open-source-ml-with-open-se-cura/`. Accessed on: Jan. 3, 2024.

[13] Antmicro. Renode. `https://renode.io`. Accessed on: Jan. 3, 2024.

[14] Antmicro. Renode documentation - describing platforms. `https://renode.readthedocs.io/en/latest/basic/describing_platforms.html`.

[15] Antmicro. renode-dpi-examples. `https://github.com/antmicro/renode-dpi-examples`. Accessed on: Jan. 5, 2024.

[16] Antmicro. renode-verilator-integration. `https://github.com/antmicro/renode-verilator-integration`. Accessed on: Jan. 5, 2024.

[17] Antmicro. renode-verilator-plugin. `https://github.com/renode/renode/tree/master/src/Plugins/VerilatorPlugin`. Accessed on: Jan. 5, 2024.

[18] Antmicro. renode-verilator-samples. `https://github.com/renode/renode/tree/master/scripts/single-node`. Accessed on: Jan. 5, 2024.

[19] Antmicro. U-boot dashboard. `https://u-boot-dashboard.renode.io`. Accessed on: Jan. 3, 2024.

[20] Antmicro. Zephyr dashboard. `https://zephyr-dashboard.renode.io`. Accessed on: Jan. 3, 2024.

[21] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805*, 2017.

[22] ARM-Software. Trusted board boot. `https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/design/trusted-board-boot.rst`, 2023.

[23] Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of trusted channels based on OpenSSL. In *STC '08*, STC '08, pages 41–50, New York, NY, USA, 2008. ACM.

[24] Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of trusted channels based on openssl. In Shouhuai Xu, Cristina Nita-Rotaru, and Jean-Pierre Seifert, editors, *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008*, pages 41–50. ACM, 2008.

[25] Sergei Arnautov, Andrey Brito, Pascal Felber, Christof Fetzer, Franz Gregor, Robert Krahn, Wojciech Ozga, André Martin, Valerio Schiavoni, Fábio Silva, Marcus Tenorio, and Nikolaus Thummel. PubSub-SGX: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems. In *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*, pages 123–132. IEEE Computer Society, 2018.

[26] Mudassar Aslam, Christian Gehrmann, and Mats Björkman. Asarp: automated security assessment & audit of remote platforms using tcg-scap synergies. *Journal of Information Security and Applications*, 22:28–39, 2015.

[27] Mudassar Aslam, Bushra Mohsin, Abdul Nasir, and Shahid Raza. Fonac-an automated fog node audit and certification scheme. *Computers & Security*, 93:101759, 2020.

[28] N. Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols, 11th International Workshop, Cambridge, UK, April 2-4, 2003, Revised Selected Papers*, volume 3364 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2003.

[29] Pierre-Louis Aublin, Florian Kelbert, Dan O'Keffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzer, David Eyers, and Peter Pietzuch. TaLoS: Secure and transparent TLS termination inside SGX enclaves. Technical report, Department of Computing, Imperial College London, 2017.

[30] AWS. Aws iot. `https://aws.amazon.com/com/iot/`.

[31] NorazahAbd Aziz, Nur Izura Udzir, and Ramlan Mahmod. Extending TLS with mutual attestation for platform integrity assurance. *J. Commun.*, 9(1):63–72, 2014.

[32] Ivo Babuška. The finite element method with lagrangian multipliers. *Numerische Mathematik*, 20(3):179–192, 1973.

[33] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

[34] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[35] Raphaël Barazzutti, Pascal Felber, Hugues Mercier, Emanuel Onica, and Etienne Rivière. Efficient and confidentiality-preserving content-based publish/subscribe with prefiltering. *IEEE Trans. Dependable Secur. Comput.*, 14(3):308–325, 2017.

[36] Stefano Berlato, Umberto Morelli, Roberto Carbone, and Silvio Ranise. End-to-end protection of IoT communications through cryptographic enforcement of access control policies. In Shamik Sural and Haibing Lu, editors, *Data and Applications Security and Privacy XXXVI - 36th Annual IFIP WG 11.3 Conference, DBSec 2022, Newark, NJ, USA, July 18-20, 2022, Proceedings*, volume 13383 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2022.

[37] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with bft-smart. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.

[38] H. Birkholz, M. Eckel, S. Bhandari, E. Voit, B. Sulzen, L. Xia, T. Laffey, and G. Fedorkow. A YANG Data Model for Challenge-Response-based Remote Attestation Procedures using TPMs. Internet-Draft draft-ietf-rats-yang-tpm-charra-05, Internet Engineering Task Force, July 2021. Standards Track.

[39] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. Remote Attestation Procedures Architecture. Internet-Draft draft-ietf-rats-architecture-02, Internet Engineering Task Force, March 2020. Work in Progress.

[40] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, 2017.

[41] Cristian Borcea, Arnab Deb Gupta, Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. PICADOR: end-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *Future Gener. Comput. Syst.*, 71:177–191, 2017.

[42] Bratus, Sergey and D'Cunha, Nihal and Sparks, Evan and Smith, Sean W. TOCTOU, Traps, and Trusted Computing. In *Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications*, Trust '08, pages 14–32, Berlin, Heidelberg, 2008. Springer-Verlag.

[43] Sébanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. How TrustZone could be bypassed: Side-channel attacks on a modern system-on-chip. In Gerhard P. Hancke and Ernesto Damiani, editors, *Information Security Theory and Practice - 11th IFIP WG 11.2 International Conference, WISTP 2017, Heraklion, Crete, Greece, September 28-29, 2017, Proceedings*, volume 10741 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2017.

[44] Erh-Chung Chen and Che-Rung Lee. Ltd: Low temperature distillation for robust adversarial training. *arXiv preprint arXiv:2111.02331*, 2021.

[45] Liqun Chen and Rainer Urian. Algorithm agility–discussion on tpm 2.0 ecc functionalities. In *International Conference on Research in Security Standardisation*, pages 141–159. Springer, 2016.

[46] Yongkang Chen, Ming Zhang, Jin Li, and Xiaohui Kuang. Adversarial attacks and defenses in image classification: A practical perspective. In *2022 7th International Conference on Image, Vision and Computing (ICIVC)*, pages 424–430. IEEE, 2022.

[47] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel tdx demystified: A top-down approach. *arXiv preprint arXiv:2303.15540*, 2023.

[48] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *CoRR*, abs/2303.15540, 2023.

[49] Abraham A Clements, Naif Saleh Almakhdhub, Saurabh Bagchi, and Mathias Payer. ACES: Automatic compartments for embedded systems. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 65–82, Baltimore, MD, August 2018. USENIX Association.

[50] CNBC. Google admits partners leaked more than 1,000 private conversations with google assistant. `https://www.cnbc.com/2019/07/11/google-admits-leaked-private-voice-conversations.html`, 2019.

[51] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux device drivers.* " O'Reilly Media, Inc.", 2005.

[52] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016.

[53] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 857–874. USENIX Association, 2016.

[54] cpp-httplib developers. cpp-httplib: A C++ header-only HTTP/HTTPS server and client library. `https://github.com/yhirose/cpp-httplib`.

[55] Georgios Damaskinos, El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Sébastien Rouault. Aggregathor: Byzantine machine learning via robust gradient aggregation. In *Conference on Machine Learning and Systems*, 2019.

[56] Yao Deng, Xi Zheng, Tianyi Zhang, Chen Chen, Guannan Lou, and Miryung Kim. An analysis of adversarial attacks and defenses on autonomous driving models. In *2020 IEEE international conference on pervasive computing and communications (PerCom)*, pages 1–10. IEEE, 2020.

[57] Mongoose developers. Mongoose. `https://wolfssl.com`.

[58] WolfSSL developers. WolfSSL. `https://mongoose.ws`.

[59] WolfSSL developers. WolfSSL for Intel SGX. `https://github.com/wolfSSL/wolfssl/tree/master/IDE/LINUX-SGX`.

[60] Advanced Micro Devices. AMD SEV-TIO: Trusted i/o for secure encrypted virtualization. `https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf`.

[61] Tobias Distler, Christopher Bahn, Alysson Neves Bessani, Frank Fischer, and Flavio Paiva Junqueira. Extensible distributed coordination. *Proceedings of the Tenth European Conference on Computer Systems*, 2015.

[62] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[63] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[64] European Cyber Security Organisation (ECSO). European Cyber Security Certification: A meta-scheme approach. Technical Report December, 2017.

[65] Eurotech. Dynagate 10-12, Mar 2021.

[66] Eurotech. Reliagate 10-12, Mar 2021.

[67] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.*, 54(6):126:1–126:36, 2022.

[68] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 287–305. ACM, 2017.

[69] Robot Framework Foundation. Robot framework. `https://robotframework.org/`. Accessed on: Jan. 29, 2024.

[70] Sarah Abdelwahab Gaballah, Christoph Coijanovic, Thorsten Strufe, and Max Mühlhäuser. 2PPS – publish/subscribe with provable privacy. In *40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, September 20-23, 2021*, pages 198–209. IEEE, 2021.

[71] Cesare Garlati and Sandro Pinto. Secure IoT Firmware For RISC-V Processors. *Embbedded world*, 2021, 2021.

[72] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In Peng Ning, Vijay Atluri, Shouhuai Xu, and Moti Yung, editors, *Proceedings of the 2nd ACM Workshop on Scalable Trusted Computing, STC 2007, Alexandria, VA, USA, November 2, 2007*, pages 30–40. ACM, 2007.

[73] GlobalPlatform. Specification library. `https://globalplatform.org/specs-library/`, 2023.

[74] Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially robust distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3996–4003, 2020.

[75] Kenneth A. Goldman, Ronald Perez, and Reiner Sailer. Linking remote attestation to secure tunnel endpoints. In Ari Juels, Gene Tsudik, Shouhuai Xu, and Moti Yung, editors, *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing, STC 2006, Alexandria, VA, USA, November 3, 2006*, pages 21–24. ACM, 2006.

[76] David Goltzsche, Manuel Nieke, Thomas Knauth, and Rüdiger Kapitza. AccTEE: A webassembly-based two-way sandbox for trusted resource accounting. In *Middleware '19*, Middleware '19, pages 123–135. ACM, 2019.

[77] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[78] Google. Cfu playground. `https://github.com/google/CFU-Playground/`. Accessed on: Jan. 3, 2024.

[79] Google. More information about our processes to safeguard speech data. `https://www.blog.google/products/assistant/more-information-about-our-processes-safeguard-speech-data/`, 2019.

[80] Google. Pub/sub, 2023.

[81] Christian Göttel, Pascal Felber, and Valerio Schiavoni. Developing secure services for IoT with OP-TEE: A first look at performance and usability. In José Pereira and Laura Ricci, editors, *Distributed Applications and Interoperable Systems - 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings*, volume 11534 of *Lecture Notes in Computer Science*, pages 170–178. Springer, 2019.

[82] Franz Gregor, Wojciech Ozga, Sébastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, André Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. Trust management as a service: Enabling trusted execution in the face

of byzantine stakeholders. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 502–514. IEEE, 2020.

[83] Trusted Computing Group. Trusted Platform Module (TPM) Summary. https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/, Feb 2021.

[84] Rachid Guerraoui, Arsany Guirguis, Jérémy Plassmann, Anton Ragot, and Sébastien Rouault. Garfield: System support for byzantine machine learning (regular paper). *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 39–51, 2021.

[85] Jessica Fitzgerald-McKay Guy Fedorkow, Eric Voit. TPM-based Network Device Remote Integrity Verification. Internet-Draft draft-ietf-rats-tpm-based-network-device-attest-06, Internet Engineering Task Force, June 2021. Informative.

[86] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

[87] IEEE. Ieee standard for verilog hardware description language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, 2006.

[88] Inclavare Containers. RATS architecture based TLS using librats. `https://github.com/inclavare-containers/rats-tls`.

[89] Intel. Introduction to Intel SGX sealing, April 2016. `https://intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html`.

[90] Intel. Overview of Intel Protected File System Library Using SGX, 2016. `https://intel.ly/34NpzMn`.

[91] Intel. Intel software guard extensions remote attestation end-to-end example, July 2018.

[92] Intel Corporation. *Intel Software Guard Extensions (Intel SGX) SDK for Linux OS — Developer Reference*, August 2023. version 2.19.

[93] Mihaela Ion, Giovanni Russello, and Bruno Crispo. Design and implementation of a confidentiality and access control solution for publish/subscribe systems. *Comput. Networks*, 56(7):2014–2037, 2012.

[94] Ed. J. Schoenwaelder. Common YANG Data Types. Internet-Draft Request for Comments: 6991, Internet Engineering Task Force, July 2013. Standards Track.

[95] Florian Jaeckle, Jingyue Lu, and M Pawan Kumar. Neural network branch-and-bound for neural network verification. *arXiv preprint arXiv:2107.12855*, 2021.

[96] André Joaquim, Miguel L. Pardal, and Miguel Correia. Vulnerability-tolerant transport layer security. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, volume 95 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[97] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: Epid provisioning and attestation services. *White Paper*, 1(1-10):119, 2016.

[98] Antanas Kascenas, Nicolas Pugeault, and Alison Q O'Neil. Denoising autoencoders for unsupervised anomaly detection in brain mri. In *International Conference on Medical Imaging with Deep Learning*, pages 653–664. PMLR, 2022.

[99] Abhirup Khanna and Rishi Anand. Iot based smart parking system. *2016 International Conference on Internet of Things and Applications (IOTA)*, pages 266–270, 2016.

[100] Anum Khurshid. Extended design and first implementation of security, safety and robustness mechanisms and tools. `https://vedliot.eu/deliverable/d5-2-extended-design-and-first-implementation-of-security-safety-and-robustness-mechanisms-and-tools/`, 2022.

[101] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. Securing real-time microcontroller systems through customized memory view switching. In *Network and Distributed System Security (NDSS) Symposium*, 2018.

[102] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *CoRR*, abs/1801.05863, 2018.

[103] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003.

[104] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[105] Murat Kuzlu, Ferhat Ozgur Catak, Umit Cali, Evren Catak, and Ozgur Guler. Adversarial security mitigations of mmwave beamforming prediction models using defensive distillation and adversarial retraining. *International Journal of Information Security*, pages 1–14, 2022.

[106] Marta Z Kwiatkowska. Safety verification for deep neural networks with provable guarantees. *30th International Conference on Concurrency Theory (CONCUR 2019)*, 2019.

[107] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *EuroSys '20*, New York, NY, USA, 2020. ACM.

[108] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 38:1–38:16. ACM, 2020.

[109] Yanlin Li, Jonathan M. McCune, James Newsome, Adrian Perrig, Brandon Baker, and Will Drewry. Minibox: A two-way sandbox for x86 native code. In Garth Gibson and Nickolai Zeldovich, editors, *ATC '14*, ATC '14, pages 409–420. USENIX, 2014.

[110] Roger A. Light. Mosquitto: server and client implementation of the MQTT protocol. *J. Open Source Softw.*, 2(13):265, 2017.

[111] Arm Limited. Exception levels. `https://developer.arm.com/documentation/102412/0103/Privilege-and-Exception-levels/Exception-levels`, 2023.

[112] Linaro Limited. Open source secure software. `https://www.trustedfirmware.org/`, 2023.

[113] Linaro. The devicetree specification. `https://www.devicetree.org/`. Accessed on: Jan. 3, 2024.

[114] Naiwei Liu, Meng Yu, Wanyu Zang, and Ravi Sandhu. On the cost-effectiveness of trustzone defense on arm platform. In *Information Security Applications: 21st International Conference, WISA 2020, Jeju Island, South Korea, August 26–28, 2020, Revised Selected Papers*, page 203–214, Berlin, Heidelberg, 2020. Springer-Verlag.

[115] Niels Lohmann. JSON for modern C++, 2023. `https://github.com/nlohmann/json`.

[116] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[117] Gavin Lowe. A hierarchy of authentication specifications. In *10th Computer Security Foundations Workshop (CSFW 1997)*, pages 31–43. IEEE Computer Society, 1997.

[118] Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Radek Fujdiak. A secure publish/subscribe protocol for internet of things. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*, pages 75:1–75:10. ACM, 2019.

[119] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. Coordination of autonomous vehicles: taxonomy and survey. *ACM Computing Surveys (CSUR)*, 54(1):1–33, 2021.

[120] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 143–158. IEEE Computer Society, 2010.

[121] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos V. Rozas. Intel Software Guard Extensions support for dynamic memory management inside an enclave. In *HASP '16*, HASP '16, pages 10:1–10:9. ACM, 2016.

[122] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification (CAV'13)*, pages 696–701. Springer, 2013.

[123] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. Attestation mechanisms for trusted execution environments demystified. In David M. Eyers and Spyros Voulgaris, editors, *Distributed Applications and Interoperable Systems: 22nd IFIP WG 6.1 International Conference, DAIS 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13272 of *Lecture Notes in Computer Science*, pages 95–113. Springer, 2022.

[124] Jämes Ménétrey, Aeneas Grüter, Peterson Yuhala, Julius Oeftiger, Pascal Felber, Marcelo Pasin, and Valerio Schiavoni. A holistic approach for trustworthy distributed systems with webassembly and tees. In *Proceedings of the 27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, 2023.

[125] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. Twine: An embedded trusted runtime for webassembly. In *ICDE*, pages 205–216. IEEE, 2021.

[126] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. WaTZ: A trusted WebAssembly runtime environment with remote attestation for TrustZone. In *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*, pages 1177–1189. IEEE, 2022.

[127] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*, 2018.

[128] Microsoft. Introducing azure iot support for x.509 certificates, Aug 2016.

[129] Microsoft. Publisher-subscriber pattern, 2023.

[130] Modberry. Modberry, Mar 2021.

[131] Nir Morgulis, Alexander Kreines, Shachar Mendelowitz, and Yuval Weisglass. Fooling a real car with adversarial traffic signs. *arXiv preprint arXiv:1907.00374*, 2019.

[132] Jämes Ménétrey. WAMR – remote attestation and secure channel of communication. `https://github.com/bytecodealliance/wasm-micro-runtime/discussions/1664`.

[133] Jämes Ménétrey. Twine runtime and experiments, 2023.

[134] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. WaTZ: A trusted WebAssembly runtime environment with remote attestation for TrustZone. In *ICDCS '22*, ICDCS '22, pages 1177–1189. IEEE, 2022.

[135] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, Giovanni Mazzeo, Arne Hollum, and Darshan Vaydia. A comprehensive trusted runtime for webassembly with intel sgx. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2023.

[136] Mohamed Nabeel, Stefan Appel, Elisa Bertino, and Alejandro P. Buchmann. Privacy preserving context aware publish subscribe systems. In Javier López, Xinyi Huang, and Ravi S. Sandhu, editors, *Network and System Security - 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings*, volume 7873 of *Lecture Notes in Computer Science*, pages 465–478. Springer, 2013.

[137] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. Everything you should know about intel SGX performance on virtualized systems. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(1):5:1–5:21, 2019.

[138] Arto Niemi, Vasile Adrian Bogdan Pop, and Jan-Erik Ekberg. Trusted sockets layer: A TLS 1.3 based trusted channel protocol. In Nicola Tuveri, Antonis Michalas, and Billy Bob Brumley, editors, *Secure IT Systems - 26th Nordic Conference, NordSec 2021, Virtual Event, November 29-30, 2021, Proceedings*, volume 13115 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2021.

[139] Arto Niemi, Sampo Sovio, and Jan-Erik Ekberg. Towards interoperable enclave attestation: Learnings from decades of academic work. In *31st Conference of Open Innovations Association, FRUCT 2022, Helsinki, Finland, April 27-29, 2022*, pages 189–200. IEEE, 2022.

[140] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. Confidentiality-preserving publish/subscribe: A survey. *ACM Comput. Surv.*, 49(2):27:1–27:43, 2016.

[141] OpenTitan. Opentitan project. `https://opentitan.org`.

[142] Nicolas Papernot and Patrick McDaniel. Extending defensive distillation. *arXiv preprint arXiv:1705.05264*, 2017.

[143] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[144] Marcelo Pasin, Jämes Ménétrey, Anum Khurshid, Alysson Bessani, Piotr Zierhoffer, and Micha vor dem Berge. D5.1 — design and early release of security, safety and robustness mechanisms and tools. Technical report, VEDLIoT, 2021.

[145] Marcelo Passin. Design and early release of security, safety and robustness mechanisms and tools. `https://vedliot.eu/deliverable/deliverable-d51/`, 2021.

[146] A Paverd. *Enhancing communication privacy using trustworthy remote entities*. PhD thesis, University of Oxford, 2015.

[147] Jinglei Pei, Yuyang Shi, Qingling Feng, Ruisheng Shi, Lina Lan, Shui Yu, Jinqiao Shi, and Zhaofeng Ma. An efficient confidentiality protection solution for pub/sub system. *Cybersecur.*, 6(1):34, 2023.

[148] Raspberry Pi. Raspberry pi 4, May 2021.

[149] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.*, 51(6), jan 2019.

[150] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. Secure content-based routing using Intel software guard extensions. In *Proceedings of the 17th International Middleware Conference, Trento, Italy, December 12 - 16, 2016*, page 10. ACM, 2016.

[151] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A. Sartakov, and Peter R. Pietzuch. SGX-LKL: securing the host OS interface for trusted execution. *CoRR*, abs/1908.11143, 2019.

[152] Ivan Puddu, Moritz Schneider, Daniele Lain, Stefano Boschetto, and Srdjan Capkun. On (the lack of) code confidentiality in trusted execution environments. *CoRR*, abs/2212.07899, 2022.

[153] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[154] Fiana Raiber and Oren Kurland. Kullback-leibler divergence revisited. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 117–124, 2017.

[155] Eric Rescorla. Keying material exporters for transport layer security (TLS). RFC 5705, March 2010.

[156] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS encrypted client hello. Internet-Draft draft-ietf-tls-esni-16, Internet Engineering Task Force, April 2023. Work in Progress.

[157] Peter Saint-Andre and Jeff Hodges. Representation and verification of domain-based application service identity within internet public key infrastructure using X.509 (PKIX) certificates in the context of transport layer security (TLS). RFC 6125, March 2011.

[158] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. Demystifying attestation in Intel trust domain extensions via formal verification. *IEEE Access*, 9:83067–83079, 2021.

[159] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[160] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[161] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. MQT-TZ: Hardening IoT brokers using ARM TrustZone : (practical experience report). In *International Symposium on Reliable Distributed Systems, SRDS 2020, Shanghai, China, September 21-24, 2020*, pages 256–265. IEEE, 2020.

[162] Carlton Shepherd, Raja Naeem Akram, and Konstantinos Markantonakis. Establishing mutually trusted channels for remote sensing devices with trusted execution environments. In *Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, August 29 - September 01, 2017*, pages 7:1–7:10. ACM, 2017.

[163] Siemens. Questa advanced simulator. `https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/`. Accessed on: Jan. 29, 2024.

[164] SpinalHDL. Vexriscv. `https://github.com/SpinalHDL/VexRiscv`. Accessed on: Jan. 4, 2024.

[165] Ron Stajnrod, Raz Ben Yehuda, and Nezer Jacob Zaidenberg. Attacking trustzone on devices lacking memory protection. *J. Comput. Virol. Hacking Tech.*, 18(3):259–269, 2022.

[166] Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In Shouhuai Xu, Cristina Nita-Rotaru, and Jean-Pierre Seifert, editors, *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008*, pages 1–10. ACM, 2008.

[167] Frederic Stumpf, Omid Tafreschi, Patrick Röder, Claudia Eckert, et al. A robust integrity reporting protocol for remote attestation. In *Proceedings of the Workshop on Advances in Trusted Computing (WATC)*, page 65, 2006.

[168] Antero Taivalsaari, Tommi Mikkonen, and Cesare Pautasso. Towards seamless IoT device-edge-cloud continuum: Software architecture options of IoT devices revisited. In Maxim Bakaev, In-Young Ko, Michael Mrissa, Cesare Pautasso, and Abhishek Srivastava, editors, *ICWE 2021 Workshops - ICWE 2021 International Workshops, BECS and Invited Papers, Biarritz, France, May 18-21, 2021*, volume 1508 of *Communications in Computer and Information Science*, pages 82–98. Springer, 2021.

[169] Team libtom. Libtomcrypt. `https://github.com/libtom/libtomcrypt`, 2023.

[170] The Linux Foundation. The zephyr project. `https://zephyrproject.org/`. Accessed on: Jan. 29, 2024.

[171] The Linux Foundation. The zephyr project twister. `https://github.com/zephyrproject-rtos/zephyr/blob/main/scripts/twister`. Accessed on: Jan. 29, 2024.

[172] Throw The Switch. Unity test project. `https://github.com/ThrowTheSwitch/Unity`. Accessed on: Jan. 29, 2024.

[173] Tate Tian. Understanding SGX protected file system, January 2017. `https://git.io/JtDP9`.

[174] Milan Tripathi. Facial image denoising using autoencoder and unet. *Heritage and Sustainable Development*, 3(2):89, 2021.

[175] Trusted Computing Group. *Trusted Platform Module Library Part 1: Architecture*, 3 2014. Level 00 Revision 01.07.

[176] Trusted Computing Group. *Trusted Platform Module Library Part 3: Commands*, 9 2016. Family 2.0, Level 00 Revision 01.38.

[177] Trusted Computing Group. *Trusted Platform Module Library Part 2: Structures*, 9 2018. Level 00 Revision 01.50".

[178] Trusted Computing Group. *DICE Attestation Architecture*, 2021.

[179] TrustedFirmware.org. Pseudo trusted applications. `https://optee.readthedocs.io/en/latest/architecture/trusted_applications.html#pseudo-trusted-applications`, 2023.

[180] European Union. European level 6g Flagship project, December 2023.

[181] Peter VanNostrand, Ioannis Kyriazis, Michelle Cheng, Tian Guo, and Robert J. Walls. Confidential deep learning: Executing proprietary models on untrusted devices. *ArXiv*, abs/1908.10730, 2019.

[182] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. COBRA: dynamic proactive secret sharing for confidential BFT services. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1335–1353. IEEE, 2022.

[183] Veripool. Verilator. `https://www.veripool.org/verilator/`. Accessed on: Jan. 29, 2024.

[184] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. Establishing secure communication channels using remote attestation with TPM 2.0. In Konstantinos Markantonakis and Marinella Petrocchi, editors, *Security and Trust Management - 16th International Workshop, STM 2020, Guildford, UK, September 17-18, 2020, Proceedings*, volume 12386 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2020.

[185] Kevin Walsh and John Manferdelli. Mechanisms for mutual attested microservice communication. In Ashiq Anjum, Alan Sill, Geoffrey C. Fox, and Yong Chen, editors, *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, UCC 2017, Austin, TX, USA, December 5-8, 2017*, pages 59–64. ACM, 2017.

[186] Robert Walther, Carsten Weinhold, and Michael Roitzsch. RATLS: integrating transport layer security with remote attestation. In Jianying Zhou, Sridhar Adepu, Cristina Alcaraz, Lejla Batina, Emiliano Casalicchio, Sudipta Chattopadhyay, Chenglu Jin, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Weizhi Meng, Stjepan Picek, Jun Shao, Chunhua Su, Cong Wang, Yury Zhauniarovich, and Saman A. Zonouz, editors, *ACNS '22*, volume 13285 of *ACNS '22*, pages 361–379. Springer, 2022.

[187] Chenxi Wang, Antonio Carzaniga, David Evans, and Alexander L Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, pages 3940–3947. IEEE, 2002.

[188] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34:29909–29921, 2021.

[189] Shuran Wang, Dahan Pan, Runhan Feng, and Yuanyuan Zhang. Magikcube: Securing cross-domain publish/subscribe systems with enclave. In *20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021, Shenyang, China, October 20-22, 2021*, pages 147–154. IEEE, 2021.

[190] WebAssembly community. WebAssembly system interface — WASI application ABI, September 2020. `https://git.io/JT3L1`.

[191] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. TIMBER-V: Tag-isolated memory bringing fine-grained enclaves to RISC-V. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[192] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. The severest of them all: Inference attacks against secure virtual enclaves. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, pages 73–85. ACM, 2019.

[193] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[194] Logan G Wright, Tatsuhiro Onodera, Martin M Stein, Tianyu Wang, Darren T Schachter, Zoey Hu, and Peter L McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022.

[195] Weigang Wu, Jiebin Zhang, Aoxue Luo, and Jiannong Cao. Distributed mutual exclusion algorithms for intersection traffic control. *IEEE Transactions on Parallel and Distributed Systems*, 26:65–74, 2015.

[196] AMD Xilinx. Vivado. `https://www.xilinx.com/products/design-tools/vivado.html`. Accessed on: Jan. 29, 2024.

[197] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020.

[198] Yue Yu, Huaimin Wang, Bo Liu, and Gang Yin. A trusted remote attestation model based on trusted computing. In *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013*, pages 1504–1509. IEEE Computer Society, 2013.

[199] Chaoqun Yue, Xiaomin Zhu, Zhiyong Liu, Xiaodong He, Zhihua Zhang, and Wei Zhao. A denoising autoencoder approach for poisoning attack detection in federated learning. *IEEE Access*, 9:43027–43036, 2021.

[200] Peterson Yuhala, Jämes Ménétrey, Pascal Felber, Marcelo Pasin, and Valerio Schiavoni. Fortress: Securing IoT peripherals with trusted execution environments. In *Proceedings of the The 39th ACM/SIGAPP Symposium On Applied Computing*, 2024.

[201] Zeuson0. Pull request – linux-sgx: Improve the remote attestation. `https://github.com/bytecodealliance/wasm-micro-runtime/pull/1695`.

[202] Piotr Zierhoffer. Second implementation of security, safety and robustness mechanisms and tools. `https://vedliot.eu/deliverable/d5-3-second-implementation-of-security-safety-and-robustness-mechanisms-and-tools-ant-pu-m27/`, 2023.