Integration of Post-Quantum Cryptography in the Keylime Agent for Quantum-Safe Remote Attestation

Integration of Post-Quantum Cryptography in the Keylime Agent for Quantum-Safe Remote Attestation

Francesco Vaccaro[1,*], Enrico Bravi[1] and Antonio Lioy[1]

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

Abstract

In recent years, Post-Quantum Cryptography (PQC) has gained considerable interest. This is mainly driven by the rapid development of Quantum Computing technologies, that are continuously improving the performance. This rapid development is threatening most of the current security methodologies, because they are largely based on classical crypto systems. For this reason, the transition from classical cryptography to PQC is becoming paramount, as the threat of quantum computers is not yet present but is expected in the very near future. This is necessary for a wide range of devices, ranging from cloud to IoT scenarios. The latter has more criticalities in performing this transition, due to the resource-constrained nature of the deployed devices. Several standardisation organisations are publishing their guideline for the adoption of PQC, based on different parameters such as the scenario and the security level. A specific security aspect that is largely affected by the threat of quantum computers is integrity verification. This is a set of techniques that aim to verify and enforce that a specific device behaves as intended. Being largely based on cryptographic operations, especially the Remote Attestation process, the transition to PQC is crucial. In this paper, we present the integration of PQC, specifically the ML-DSA algorithm, into the Keylime framework for remote attestation. In particular, we present the integration of PQC in the Keylime Agent, which is the most crucial component, as it is deployed in untrusted environments and on various devices with differing computational resources. We also propose a deployment architecture based on the Arm TrustZone TEE technology, exploiting the OP-TEE framework as trusted operating system, integrating an ML-DSA-enabled firmware TPM implementation for PQ remote attestation.

Keywords

Remote Attestation, Keylime, Post Quantum, Quantum Resistant, OP-TEE

1. Introduction

In today's digital landscape, ensuring the integrity and authenticity of computing systems is paramount; therefore, integrity verification techniques are crucial features of a secure computing environment [1]. Integrity Verification encompasses mechanisms such as secure boot, which prevents unauthorised code from executing during startup, measured boot, which records the integrity state of each component before its execution during the boot process, and runtime measurement, which ensures the system remains unaltered during its operation. Remote Attestation, on the other hand, allows a system to provide verifiable proof of its integrity status to a remote entity, facilitating trust in distributed Internet of Things (IoT) [2] and cloud-based [3] environments. Together, these mechanisms form a robust defence against unauthorised modifications and intrusions. The necessity for these security measures stems from the increasing sophistication of cyber threats. Attackers continuously devise new methods to compromise systems by altering boot sequences, injecting malicious code during runtime, or altering system configurations. The advent of the Cryptographically Relevant Quantum Computer (CRQC) poses a significant challenge to the validity of Integrity Verification mechanisms, which widely use cryptographic algorithms such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC), as they are threatened by the computational capabilities of quantum computers. As quantum technology advances, the urgency to transition to quantum-resistant cryptographic procedures becomes increasingly evident, particularly with regard to the mechanisms that underpin trust in modern

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

*Corresponding author.

†These authors contributed equally.

francesco.vaccaro@polito.it (F. Vaccaro); enrico.bravi@polito.it (E. Bravi); antonio.lioy@polito.it (A. Lioy)

0009-0008-0222-5365 (F. Vaccaro); 0009-0006-1832-0274 (E. Bravi); 0000-0002-5669-9338 (A. Lioy)

digital infrastructures. Migrating Integrity Verification techniques to quantum-safe approaches involves selecting cryptographic algorithms that can withstand the capabilities of CRQCs, and integrating them in secure boot, measured boot, runtime integrity, and Remote Attestation (RA) processes. This demands a comprehensive overhaul of key management practices, cryptographic libraries, and protocol standards used in existing Integrity Verification frameworks. The transition of Trusted Computing procedures in response to the imminent revolution brought about by the application of quantum computing is a complex exercise that must be addressed urgently, as it is considered a priority by the scientific community compared to other security techniques. In this paper, we propose the integration of Post-quantum cryptography (PQC) in the Keylime Agent. Keylime can be considered the standard *de facto* framework for RA. In particular, its initial focus was primarily on the cloud environment, but it has recently begun developing to support the needs of embedded and IoT devices. We propose integrating the ML-DSA algorithm, considering the specific scenario of continuous RA, which has specific performance requirements for cryptographic operations.

This paper has the following structure. In Section 2 background concepts are presented, focusing on PQC and Trusted Computing aspects. In Section 3 is presented the integration of the ML-DSA algorithm in the Keylime Agent, while in Section 4 the implementation aspects are presented. Finally, in Section 6 the conclusions are described, and future works are proposed.

2. Background

2.1. Post-Quantum Cryptography

PQC consists of a family of cryptographic algorithms designed to be resistant to quantum computers, whose capabilities threaten the hardness assumptions underlying widely deployed public-key systems such as RSA and ECC. Different from classical cryptography, PQC does not rely on integer factorisation or discrete logarithms, but on mathematical problems believed to be challenging to solve even in the presence of large-scale quantum computers. Among the various families, Lattice-based cryptography has emerged as one of the most prominent due to its strong theoretical foundations, built on problems such as Learning With Errors (LWE), Ring-LWE and Module-LWE, and its efficient performance on hardware. Algorithms like CRYSTALS-Kyber [4] (key encapsulation) and CRYSTALS-Dilithium [5] (digital signatures) exemplify this approach, offering compact keys, fast operations, and high assurance against structural attacks. Another important category is Hash-based cryptography, which associates a one-time signature scheme, i.e. Lamport signature, with a Merkle tree structure. The most relevant algorithm in this category is SPHINCS+ [6], a stateless hash-based signature scheme that allows creating multiple instances depending on the hash function, the type (f/s for time/size optimisation), and the security level. Multivariate cryptography, on the other hand, relies on the difficulty of solving systems of multivariate polynomials over finite fields, which is proven to be NP-complete. Most of the algorithms proposed in this category have polynomials of degree two; thus, we usually reduce the inspection to MQ (multivariate-quadratic) problems. Multivariate cryptography is preferred for signature schemes because it is known to produce the shortest signatures among PQC algorithms. Similarly, Code-based cryptography (CBC) involves the NP-complete problem of decoding a random linear code. CBC is founded on hard problems from algebraic coding theory (i.e. error-correcting codes). The last relevant family is Isogeny-based cryptography, which uses the difficulty of finding isogenies between supersingular elliptic curves. A notable algorithm in this family is the Supersingular Isogeny Key Encapsulation (SIKE), which corresponds to the quantum case of the DH key exchange. Although initially attractive due to its small key sizes, SIKE was broken in 2022 (its attack published in [7]) and later removed from the list of fourth-round candidate algorithms by NIST.

The National Institute of Standards and Technology (NIST) plays a major role in evaluating and standardising post-quantum cryptographic schemes. In 2016, the NIST PQC project issued a call for proposals for quantum-resistant digital signature and key encapsulation mechanisms, starting several rounds of public review and comment. In 2024, after an eight-year process, three Federal Information Processing Standards were standardised: FIPS 203 (based on CRYSTALS-Kyber, which

FIPS 203 - ML-KEM (Lattice-based Key-Encapsulation)				
Algorithm	encapsulation key	decapsulation key	ciphertext	shared secret key
ML-KEM-512	800	1,632	768	32
ML-KEM-768	1,184	2,400	1,088	32
ML-KEM-1024	1,568	3,168	1,568	32

Table 1
Sizes (in bytes) of keys and ciphertext of ML-KEM [8].

FIPS 204 - ML-DSA (Lattice-based Signatures)				
Algorithm	Security category	Private key	Public key	Signature
ML-DSA-44	2	2,560	1,312	2,420
ML-DSA-65	3	4,032	1,952	3,309
ML-DSA-87	5	4,896	2,592	4,627
FIPS 205 - SLH-DSA (Stateless Hash-Based Signatures)				
Algorithm	Security category	Private key	Public key	Signature
SLH-DSA-128s	1	64	32	7,856
SLH-DSA-192s	3	96	48	16,224
SLH-DSA-256s	5	128	64	29,792

Table 2
Sizes (in bytes) of keys and signatures of ML-DSA [9] and SLH-DSA [10].

has been renamed ML-KEM), FIPS 204 (based on CRYSTALS-Dilithium, renamed ML-DSA) and FIPS 205 (based on SPHINCS+, renamed SLH-DSA). A summary of these three algorithms is available in Table 1 and Table 2. However, many other international bodies are focusing on PQC development, including the European Telecommunications Standards Institute (ETSI), the International Organization for Standardization (ISO), and the Internet Engineering Task Force (IETF). Furthermore, it is essential to highlight open-source initiatives, such as the Open Quantum Safe (OQS) project [11], which aims to facilitate the transition towards a quantum-resistant world. Through libraries like `liboqs` [12], researchers and developers can evaluate, benchmark, and compare PQC schemes within existing protocols. Tools like `liboqs` reveal that the algorithm's choice often depends on the application context: lattice-based schemes provide strong general-purpose performance; code-based schemes remain appealing for long-term archival security; and hash-based signatures function well in systems prioritising minimal trust assumptions. As quantum technologies progress, the diversity of PQC primitives ensures that organisations can tailor cryptographic migration plans to their performance constraints, security requirements, and long-term interoperability needs.

2.2. Trusted Computing and Remote Attestation

Securing modern infrastructures is becoming paramount due to their more critical purposes. Trusted Computing provides crucial security techniques in this direction, that aim to build trust in platforms. Accordingly with the Trusted Computing Group (TCG) a component is *trusted* when it is always expected to behave as intended [13]. This property has been achieved through the definition of the concept of Root of Trust (RoT), a component that expose a set of trustworthy functions that the system can use to enforce strong levels of security.

1. *Root of Trust for Measurement (RTM)*: an immutable component responsible for securely acquiring and recording integrity measurements (e.g., cryptographic hashes) of the initial mutable software components that assume control of the platform during the boot-up phase;

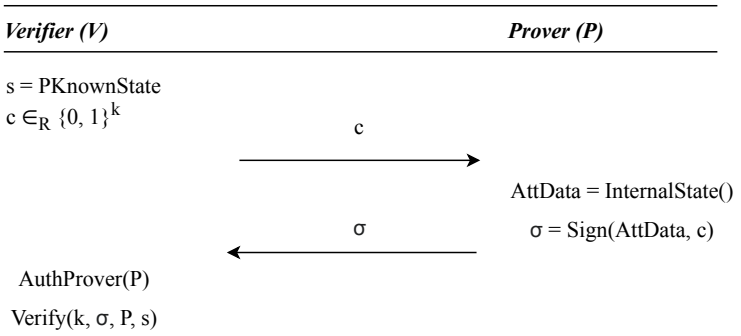

Figure 1: Remote Attestation process description [14].

2. *Root of Trust for Storage (RTS)*: a securely protected memory region designated to reliably store integrity measurements obtained by the RTM, ensuring these data remain tamper-proof throughout the device's operational lifecycle;
3. *Root of Trust for Reporting (RTR)*: a device component responsible for generating unforgeable and non-repudiable integrity reports. This element typically employs asymmetric cryptographic signatures on the measurements stored in the RTS, using cryptographic keys uniquely bound to the device's identity.

RA is a protocol used to verify the integrity and trustworthiness of a remote computing system. To enable this, the remote platform must provide mechanisms that allow a trusted party (typically referred to as *verifier*) to attest the actual state of the remote system (typically called *attester* or *prover*) in a way that is reliable even if the platform has been compromised.

To be reliable RA requires that all three fundamental RoTs are enabled within the device: RTM, RTS, and RTR. It is a challenge-response protocol (Figure 1) in which the verifier sends a challenge (c) to the attester, which then generates and signs an attestation evidence (σ) containing the requested measurements (*AttData*), the nonce, and other relevant data (e.g., timestamp, counters). The nonce is crucial in preventing replay attacks, where the attester sends the verifier non-fresh attestation evidence created before the system was compromised. Once the attestation evidence is received, the verifier can verify the signature and extract the integrity state. Then it compares the received evidence with the expected attester state (*PKnowSate*) and verify its trustworthiness.

2.2.1. Trusted Platform Module

The Trusted Platform Module (TPM) is a specification proposed by the TCG to build a Hardware RoT [15]. To have the highest level of security, a tamper-resistant chip is usually implemented that can store cryptographic keys and perform cryptographic operations. One of the key points introduced in the TPM 2.0 version [15] is the introduction of crypto-agility, in contrast to the previous 1.2 version [16], where the cryptographic operations were fixed.

The TPM specification permits the implementation of a RTS and RTR. In particular, a crucial definition for implementing the RTS is the Platform Configuration Register (PCR). PCRs are a set of registers that store digest values, and their length is fixed to the length of the output of the associated hash algorithm. The specification defines a set of 24 PCRs, referred to as a bank, for each supported hash algorithm. These registers can only be reset by a platform reset, and they do not support the write operation but only the *extend* operation. This operation allows saving a new digest in the PCR by hashing the concatenation of the old PCR value and the new digest:

$$PCR_{new} = Hash_{algo}(PCR_{old} \,||\, new\_digest)$$

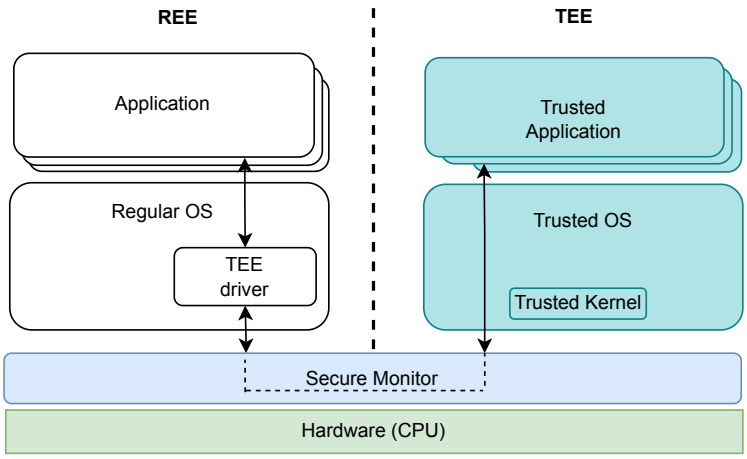

Figure 2: Trusted Execution Environment (TEE) reference architecture.

The TPM also implements the RTR by providing the `Quote` command. This command is designed to provide the values stored in the PCRs, signed with a specific Attestation Key (AK), that permits binding them with the specific TPM. An AK is a signing key that can only be used to sign data generated by the TPM. The key pair can be generated at any time within the TPM, and the private part never leaves the TPM.

The root credential managed by the TPM is the Endorsment Key (EK). The EK is an asymmetric key pair that is unique for each TPM and identifies it. In the device memory, the certificate associated with the public part of the EK is saved, which is used to authenticate and verify the validity of the TPM.

2.3. Trusted Execution Environments

TEEs [17] are a set of technologies that aim to provide a protected and shielded environment of execution for critical applications that need strong confidentiality and integrity over the code and the data. They are typically implemented as extensions of the Instraction Set Architecture (ISA) of Central Processing Units (CPUs), providing hardware-level isolation. Different TEE technologies have been proposed, mainly from industries, such as Intel Software Guard Extensions (SGX) [18], Intel Trust Domain Extensions (TDX) [19], Arm TrustZone [20], Arm Confidential Compute Architecture (CCA) [21], and AMD Secure Encrypted Virtualization (SEV) [22]. The main issue with these technologies is that, being developed by industries, they are closed-source and not customizable. For this reason, academia started working on providing TEEs that can be customized and modified based on the user requirements. Some examples of this philosophy are Keystone [23] and Sanctum [24].

A TEE divides a platform into two separate execution environments: the Rich Execution Environment (REE) and the TEE (Figure 2). The REE is the non-protected part of the platform, which serves as the common execution environment where non-critical Operating System (OS) and applications run. TEEs can additionally be divided into two main categories, based on their design and security model. The first category is the Trusted World-based TEEs (e.g., Arm TrustZone). These technologies are designed to provide two isolated execution environments, the REE and the TEE. In this case, there is no isolation among the applications running in the TEE (Trusted Application). They run protected from the rich environment, but they can eventually interfere with each other. The Enclave-based TEEs (e.g., Keystone) instead do not provide a unique TEE, but each Trusted Application gets its isolated and protected environment (commonly called *enclave*), where the application runs, together with its own trusted OS instance. In this case, the Trusted Applications are not only isolated from the REE applications, but also from each other.

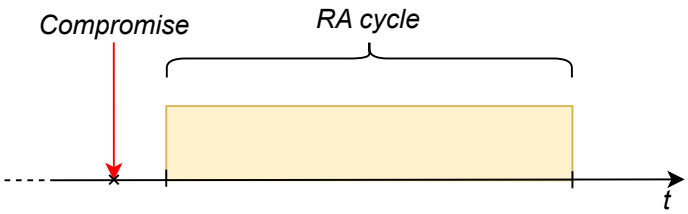

Figure 3: Impact of the RA cycle duration with regard to a compromise.

2.4. Keylime

Keylime [25] is a widely adopted RA framework based on TPM. The purpose of this project is to provide a full set of components and functionalities to attest the trustworthiness of an infrastructure. Keylime was originally proposed for trust monitoring cloud nodes, but the development team recently decided to expand the applicability to edge and IoT nodes [26].

Keylime utilises the TPM as hardware RoT to provide a reliable and secure RA process. In particular, to perform runtime attestation, it leverages the Integrity Measurement Architecture (IMA) Linux Security Module (LSM) [27]. IMA maintains a measurement log of all critical operations performed on the platform, including binary executions and file reads, which is bound to a specific PCR in the TPM (PCR 10 by default). The format of this log utilises a template mechanism that permits customisation based on the necessity to include specific information [28]. To bind the measurement log to the TPM, each time an event occurs, the corresponding measurement is extended in the PCR 10. In this way, it is possible to protect the integrity of the log, because on the verifier side, during the RA procedure, using the log, the PCR 10 value received can be recalculated. In this way, if the recalculated value does not match the received one, it means that the log received has been tampered with and, therefore, cannot be considered trustworthy. If the recomputed PCR value matches the received one, the verifier can use the log to verify that only accepted events have occurred on the platform. To perform this verification, the verifier requires a set of reference values (*whitelist*) that describe the allowed events.

3. Design Description and Considerations

3.1. Initial Analysis

To make the protocol resistant to replay attacks in a Post Quantum (PQ) scenario, the nonce must be unpredictable and infeasible to forge, even for an adversary equipped with a CRQC. Thus, the verifier must generate the nonce by relying on a quantum-resistant Pseudo Random Number Generator (PRNG) based on string properties to be resistant to quantum attacks. When the attesting system receives the attestation challenge from the verifier, it generates an attestation evidence, or quote, which serves as proof that the device's current state (including its firmware, software, and configuration) is trustworthy and has not been tampered with. The key elements of a quote are the integrity measurements performed on the system components, the nonce received from the verifier, and the signature over both the measurements and the nonce, ensuring integrity and authenticity. The nonce must be generated by the verifier according to the guidelines described above, and the signature must be performed using a PQ digital signature algorithm. In the case of PQ RA, the selected signature algorithm has to be efficient in all its operations, especially for signature generation and signature verification. The efficiency of these operations is crucial to quickly detect compromises (Figure 3) and to maintain overall system performance: the faster the attestation process, the shorter the window of opportunity for attackers to exploit a compromised system.

Considering the performance analysis [29, 30] and the TCG work on introducing PQ algorithm in the TPM specification [31], we chose to integrate ML-DSA, specifically ML-DSA-87, in order to obtain

the best solution in terms of key size and signature generation/verification time.

3.2. Platform Design

Our implementation leverages a firmware Trusted Platform Module (fTPM) running in the *secure world*. In this case, the hardware protection is provided not by a tamper-resistant discrete chip, but by the TrustZone TEE technology. In particular, the fTPM performs cryptographic signatures over PCRs that contain integrity measurements, providing trusted evidence of the device's state. To enable the transition towards PQ RA, we use a fTPM implementation which integrates the ML-DSA-87 signature algorithm [32]. Further technical details on the PQ fTPM implementation and the TPM Software Stack (TSS) extension are provided in Section 4.

We used the attestation of integrity at boot-time, enabling also continuous runtime attestation. Runtime attestation ensures ongoing verification of dynamically loaded software components within the Linux operating system, detecting unauthorised modifications and ensuring continuous integrity throughout operation. To achieve runtime attestation, we leveraged the IMA module within the Linux kernel [27, 33]. IMA, when properly configured, acquires integrity measurements from dynamically loaded kernel modules, user-space applications, libraries, and configuration files during runtime, typically extending these measurements into PCR 10. However, modifications were necessary to adapt IMA for enabling remote attestation based on the fTPM, as the default IMA implementation determines TPM availability exclusively at kernel initialization time, initializing the internal data structures to bypass the extension in the PCR if a TPM is not detected. The architectural features of Open Portable TEE (OP-TEE) cause the fTPM to be started only after the Linux kernel has completed initialisation; Hence, the default IMA behaviour required modifications to support dynamic reconfiguration. We adapted the IMA module to dynamically detect the presence of the fTPM and reconfigure its internal data structures accordingly, thus ensuring proper extension of runtime integrity measurements.

Finally, to enable remote attestation operations, we deployed the Keylime framework [25], comprising an Attestation Agent operating within the *normal world* and a remote verifier. Keylime was specifically extended to support TPM quotes using the ML-DSA-87 algorithm, enabling robust PQ remote attestation of the IoT device.

4. Implementation

For our implementation, we decided to select ML-DSA instead of other PQ algorithms, such as SLH-DSA, because it provides the best trade-off between the speed for generating the signature and its length. This also provides a good compromise between a small attack surface and a manageable signature length. To give an idea, the signature size of SPHINCS+-256f-simple is 49,856 bytes, which can be problematic in IoT devices with limited resources. Specifically, the ML-DSA-87 algorithm has been integrated, providing the highest level of security [9].

To deploy and test the final implementation, we designed a platform architecture suitable for IoT and embedded devices. This architecture comprises all the components modified to support the ML-DSA-87 algorithm, and the components needed to provide the security properties required to TPM-based RA, but considering the resource-constrained nature of the target devices.

4.1. Platform Description

To implement and deploy the final integration, we design the platform architecture as depicted in Figure 4. Considering the target scenario of IoT and embedded devices, it is impractical to opt for a physical TPM. The adoption of a fTPM in our approach is motivated by the absence of commercially available physical TPMs that provide PQ security. In addition, no PQC-integrated TPM hardware implementation is available, as the specification update has not yet been finalized [31]. Nevertheless, fTPMs remain vulnerable to firmware-level attacks and do not offer the same degree of isolation as discrete hardware TPMs. However, once post-quantum–capable hardware TPMs become available, the

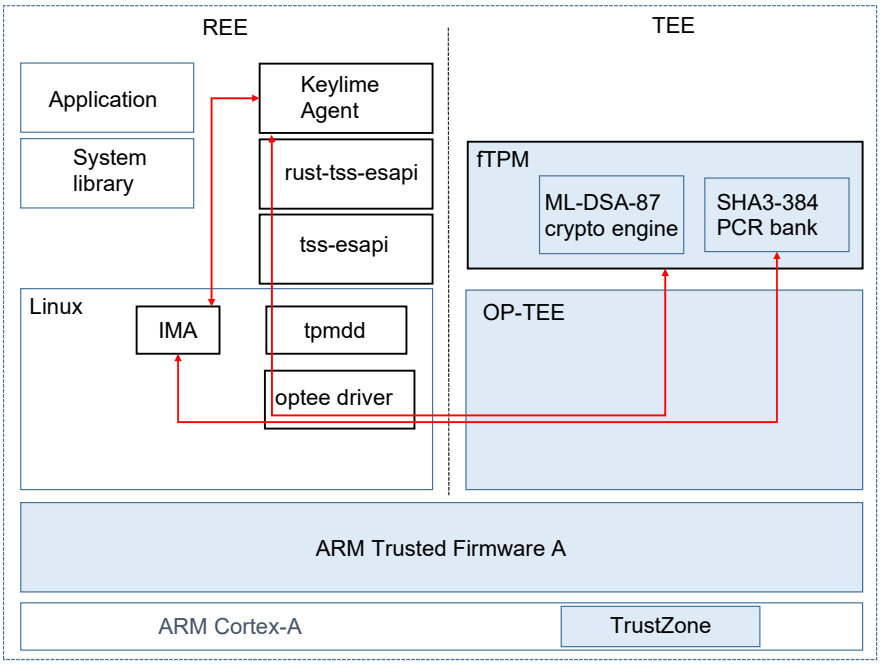

Figure 4: Defined device architecture.

integration of post-quantum mechanisms will be facilitated, as the software developed for the fTPM will already be in place and reusable. For these reasons, we designed the platform based on an Arm Cortex-A CPU, TrustZone-enabled. In this way, it is possible to provide strong security for the RTS and the RTR as recommended by the TCG [34]. In particular, regarding the RTS, the SHA3-384 algorithm was enabled for storing the measurement collected inside the PCRs. The choice of SHA3-384 is motivated by the robustness of the SHA3 family of hash functions, which is currently considered quantum-resistant [35]. Specifically, its security relies on the difficulty of finding hash collisions or preimages, for which no efficient quantum algorithms are known. Likewise, for the RTR, the quote of the fTPM was expanded to support ML-DSA-87 algorithm for the generation of the signature. This algoritms claims to meet the security level 5, a number associated with the strength of a PQ cryptographic algorithm, as specified by NIST.

To deploy the fTPM as a Trusted Application in the TEE (*Secure World*), we integrated the OP-TEE trusted OS. In particular, OP-TEE allows starting some specific applications (early trusted applications) during the Secure World boot procedure, which happens before the REE (*Normal World*) boot. In this way, it is possible to have the fTPM available during the Linux boot procedure, allowing it to collect all the measurements collected by IMA.

The choice of a suitable trusted OS fell on OP-TEE, an open-source project, developed by Linaro for the Arm TrustZone technology. It implements the APIs defined in the GlobalPlatform API specifications [36]. GlobalPlatform is an industry consortium that aims to identify and develop standards to promote secure and interoperable deployment and management of embedded hardware on secure chip technology.

4.2. Trusted Software Stack Implementation

The TSS is designed as an interface for the TPM, abstracting the low-level details and the specific TPM implementation. The TSS consists of multiple layers, allowing scalable implementations to be tailored to fit well in high-end systems and resource-constrained low-end systems. This software stack provides various APIs that allow the application to access TPM functionality. As shown in Figure 5, in the TSS each layer is responsible for a specific aspect of TPM communication. It is composed of the following components:

1. Feature API (FAPI): this is the highest level of APIs [38], which exposes the high-level functionalities of the TPM (e.g., sealing, unsealing). It reduces the complexity in performing complex

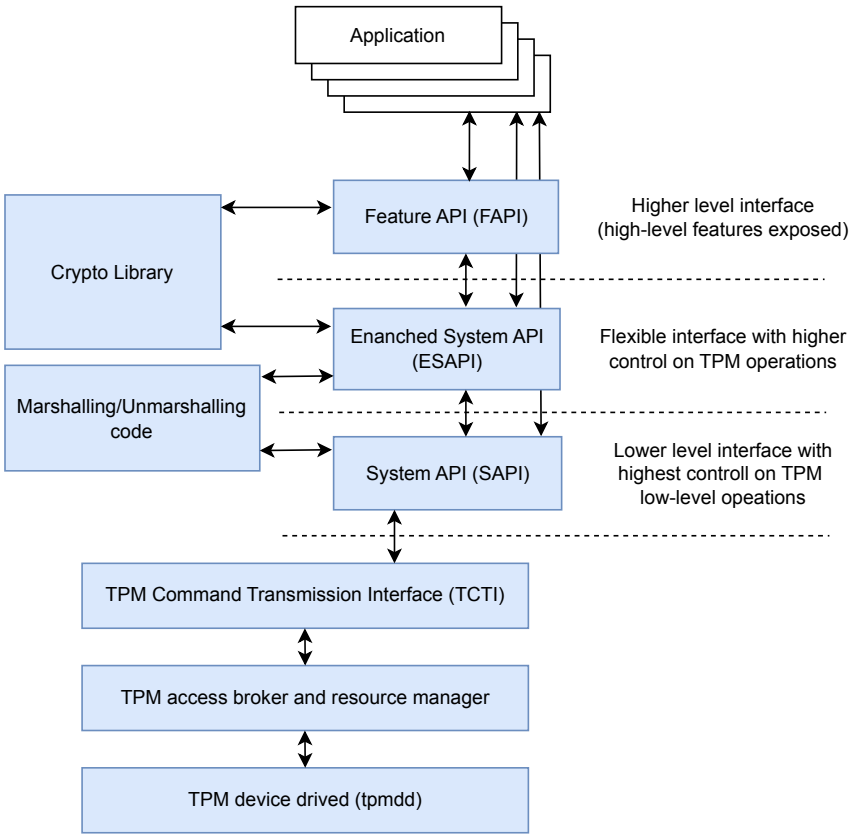

Figure 5: TPM Software Stack [37].

operations, but also the flexibility.

2. Enhanced System API (ESAPI): it is a set of APIs [39] more flexible than FAPI, providing still high-level interfaces, while enabling a high degree of customization in how commands and operations are composed.
3. System API (SAPI): this is the most flexible layer of APIs [40], which provides a low-level set of interfaces to enable the composition of all the possible combinations of input and outputs.
4. Marshalling/Unmarshalling code: this layer is responsible for composing and decomposing commands sent and received from the TPM [41]. It can be invoked from both the SAPI and the ESAPI.
5. TPM Command Transmission Interface (TCTI): this is the interface that directly sends and receives marshalled commands from the TPM (or the lower layer responsible for direct communication with the TPM) [42]. It permits interaction with different instances from the same system (e.g., hardware TPM, firmware TPM, virtual TPM).
6. TPM access broker and resource manager: The TPM access broker is responsible for managing concurrent access to the device and handling multiple requests. The TPM resource manager acts as a virtual memory manager, swapping in and out the contexts of the various active TPM interactions [43].
7. TPM device driver (tpmdd): it is the component that sends and receives data to and from the TPM. It runs in kernel mode and is vendor and OS specific.

The official implementation of the TCG specification is available in the `tpm2-tss` [44] Github repository, provided by the TPM2-software community. The community is committed to developing projects that support TSS, such as the `tpm2-tools`, a command-line interface that leverages the ESAPI interface.

To enable the operations required by Keylime to operate with ML-DSA-87, we integrated the support in the ESAPI layer. In particular, the main integration concerns key creation operations, needed to obtain the EK information for verifying the authenticity of the TPM, and to create the AK for the quote generation.

The integration of the ML-DSA-87 algorithm into the Keylime Agent brought us to various modifications into the TSS, in particular, the ESAPI layer was the most involved one. Since the Keylime Agent is written in Rust, we needed to modify the `rust-tss-esapi` wrapper [45] to support PQ operations into the TPM. The wrapper is part of the PARSEC project [46] an open-source initiative to provide a common API to secure services in a platform-agnostic way.

5. Tests and Evaluation

For the evaluation environment we used QEMU, in order to create a virtualized environment that emulates an ARMv7 board. The setup used [47] enables the execution of OP-TEE and a fTPM within the TEE, simulating conditions as closely as possible to those on real hardware. All the tests were performed on a single physical machine, consisting in a Huawei Matebook 14 machine with the following specifications:

- CPU: AMD Ryzen 5 4600H;
- RAM: 16GB DDR4;
- OS: Ubuntu 22.04.5 LTS.
- OP-TEE: version 4.5.0.
- Keylime Agent: version 0.2.7.
- TSS: version 3.3.2

To evaluate the impact of integrating ML-DSA-87 into the Keylime agent, we measured the time required for the modified cryptographic operations. We compared the results with the current supported algorithms (RSA-2048, ECDSA P-256) to highlight the potential impact. The operations we evaluated are the EK generation, the AK generation, and the quote generation.

Algorithm	Private key (B)	Public key (B)	AK gen. (s)	EK gen. (s)
RSA-2048	256	256	15.97	12.53
ECDSA P-256	32	64	12.72	9.86
ML-DSA-87	4896	2592	**11.86**	**10.21**

Table 3
Time necessary for key generation (AK and EK) with the fTPM supported algorithms.

In Table 3, the results of key generation are reported in terms of time. It is possible to notice that the time of the operations performed with ML-DSA-87 do not introduce overhead compared with the same operations performed with RSA-2048 and ECDSA P-256. This proves that creating PQ keys does not introduce any additional latency in the Keylime agent setup process. The main issue can be attributed to the key size, as in the case of ML-DSA-87, where the increase compared to RSA-2048 and ECDSA P-256 is respectively one order and two orders of magnitude. This can have an impact on future TPM hardware implementations due to the memory limitations of these devices.

The second operation we evaluated is the TPM quote. In this case, being an operation performed continuously during the RA procedure, introducing a high overhead can significantly impact the detection time of potential compromises. In Table 4, we reported the results in terms of time for generating the quote. Also, in this case, using ML-DSA-87 does not introduce any overhead compared with the currently supported algorithms. As the key generation procedure, the main impact is on the

Algorithm	Private key (B)	Public key (B)	Quote size (B)	Quote time (s)
RSA-2048	256	256	256	0.72
ECDSA P-256	32	64	64	0.28
ML-DSA-87	4896	2592	4627	**0.72**

Table 4
Time necessary for quote generation with the fTPM supported algorithms.

size of the generated quote. In this case, however, the impact is not as significant as in the key generation case, because for this operation, the overhead is not on the TPM memory, but on the platform memory.

It is possible to notice that the integration of PQC does not impact the time performance during the RA procedure. Considering the NIST guidelines [35], the migration to PQC should be performed before 2035, marking quantum-vulnerable algorithms, such as ECDSA and RSA, as *disallowed* after that date. Our results demonstrate that this migration can be implemented for RA, particularly for IoT devices, given the significant increase in security level.

6. Conclusion and Future Work

Quantum technologies are rapidly evolving, posing a concrete threat to currently adopted cryptographic algorithms and implementations. This is having a significant impact on many scenarios due to the widespread adoption of cryptography for security purposes. PQC is attempting to address this issue by introducing quantum-resistant algorithms that can replace the commonly adopted ones, thereby mitigating the risk of quantum computer attacks. Among the scenarios requiring a timely migration to PQC, RA is particularly critical, as its security guarantees strongly depend on the robustness of the underlying cryptographic primitives. In this paper, we propose the migration of the RA Keylime agent to PQC, specifically integrating the ML-DSA-87 algorithm support. We propose an IoT platform based on the Arm TrustZone TEE to deploy a PQC-enabled firmware implementation of the TPM. The decision to adopt a firmware-based TPM stems from the current unavailability of commercial hardware TPMs that support quantum-resistant cryptographic algorithms. The objective of this work is to proactively initiate a software-based implementation, thereby enabling early experimentation and validation of post-quantum mechanisms. This approach is intended to facilitate a smoother transition and immediate adoption once post-quantum–capable hardware TPMs become available on the market. The integration proved to add no overhead for the main cryptographic operations in terms of time. The only issue is related to the dimensions of the keys and the quote generated by the TPM. During this work, we experienced the classical problem of integrating existing heterogenous frameworks inside a unique coherent system. The challenges encountered highlight once more the urgency of initiating the transition to PQ at an early stage. Future work will explore the integration of additional PQ algorithms and parameters sets, enabling flexible security configurations suited to different deployment requirements.

Acknowledgments

This work has been developed within the QUBIP project (https://www.qubip.eu), funded by the European Union under the Horizon Europe framework programme [grant agreement no. 101119746]. This work was also partially supported by the SERICS project [PE00000014] under the NRRP MUR program, funded by the European Union-NextGenerationEU. This paper is also part of the project PNRR-NGEU which has received funding from the MUR-DM 352/2022.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] E. Bravi, S. Sisinni, L. Ferro, V. Donnini, A. Lioy, Implementation of the TCG DICE Specification Into the Keystone TEE Framework, IEEE Access 13 (2025) 142284–142303. doi:DOI 10.1109/ACCESS.2025.3596829.

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

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

[4] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. yubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber. Algorithm Specifications And Supporting Documentation (Version 3.02), 2021. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[5] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Dilithium. Algorithm Specifications and Supporting Documentation (Version 3.1), 2021. https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[6] J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, B. Westerbaan, Sphincs+. submission to the nist post-quantum project, v.3.1, 2022. https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[7] W. Castryck, T. Decru, An Efficient Key Recovery Attack on SIDH, in: Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon (France), April 23-27, 2023, pp. 423–447. doi:DOI 10.1007/978-3-031-30589-4_15.

[8] NIST, FIPS 203. Module-Lattice-Based Key-Encapsulation Mechanism Standard, 2024. doi:DOI 10.6028/NIST.FIPS.203.

[9] NIST, FIPS 204. Module-Lattice-Based Digital Signature Standard, 2024. doi:DOI 10.6028/NIST.FIPS.204.

[10] NIST, FIPS 205. Stateless Hash-Based Digital Signature Standard, 2024. doi:DOI 10.6028/NIST.FIPS.205.

[11] The Open Quantum Safe Project, 2026. https://openquantumsafe.org/.

[12] Open Qauntum Safe, The LibOQS Project, 2026. https://github.com/open-quantum-safe/liboqs.

[13] Trusted Computing Group, TCG Glossary, 2017. https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf.

[14] E. Bravi, D. G. Berbecaru, A. Lioy, A Flexible Trust Manager for Remote Attestation in Heterogeneous Critical Infrastructures, in: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, Naples, (Italy), 2023, pp. 91–98. URL: https://doi.org/10.1109/CloudCom59040.2023.00027. doi:DOI 10.1109/CLOUDCOM59040.2023.00027.

[15] TCG, Trusted Platform Module Library Part 1: Architecture, 2024. https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-1.83-Part-1-Architecture.pdf.

[16] TCG, TPM Main Part 1 Design Principles, 2011. https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf.

[17] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted execution environment: What it is, and what it

is not, in: 2015 IEEE Trustcom/BigDataSE/ISPA, volume 1, Helsinki (Finland), 2015, pp. 57–64. doi:DOI 10.1109/Trustcom.2015.357.

[18] V. Costan, S. Devadas, Intel SGX Explained, Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086.

[19] P. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, J. Bottomley, Intel TDX Demystified: A Top-Down Approach, ACM Computing Surveys 56 (2024) 238:1–238:33. doi:DOI 10.1145/3652597.

[20] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, S. Martin, TrustZone Explained: Architectural Features and Use Cases, in: IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Pittsburgh (PA,USA), 2016, pp. 445–451. doi:DOI 10.1109/CIC.2016.065.

[21] Arm Limited, Confidential Compute Architecture, 2026. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture.

[22] A. Sev-Snp, Strengthening VM isolation with integrity protection and more, White Paper, January 53 (2020) 1450–1465. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[23] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, D. Song, Keystone: an open framework for architecting trusted execution environments, in: EuroSys '20: Fifteenth European Conference on Computer Systems, Heraklion (Greece), 2020, pp. 1–16. doi:DOI 10.1145/3342195.3387532.

[24] V. Costan, I. Lebedev, S. Devadas, Sanctum: Minimal Hardware Extensions for Strong Software Isolation, in: 25th USENIX Security Symposium, Austin (TX, USA), 2016, pp. 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[25] N. Schear, P. T. Cable, T. M. Moyer, B. Richard, R. Rudd, Bootstrapping and maintaining trust in the cloud, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC '16, Association for Computing Machinery, Los Angeles (CA, USA), 2016, p. 65–77. URL: https://doi.org/10.1145/2991079.2991104. doi:DOI 10.1145/2991079.2991104.

[26] Keylime, The Rust Keylime Agent, 2026. https://github.com/keylime/rust-keylime.

[27] R. Sailer, X. Zhang, T. Jaeger, L. van Doorn, Design and Implementation of a TCG-based Integrity Measurement Architecture, in: 13th USENIX Security Symposium (USENIX Security 04), USENIX Association, San Diego (CA, USA), 2004. https://www.usenix.org/conference/13th-usenix-security-symposium/design-and-implementation-tcg-based-integrity-measurement.

[28] IMA Event Log Format, 2026. https://ima-doc.readthedocs.io/en/latest/event-log-format.html#ima-event-log-format.

[29] M. Kumar, Post-quantum cryptography Algorithm's standardization and performance analysis, Array 15 (2022) 100242. URL: https://doi.org/10.1016/j.array.2022.100242. doi:DOI 10.1016/J.ARRAY.2022.100242.

[30] M. Raavi, S. Wuthier, P. Chandramouli, Y. Balytskyi, X. Zhou, S. Chang, Security Comparisons and Performance Analyses of Post-quantum Signature Algorithms, in: K. Sako, N. O. Tippenhauer (Eds.), Applied Cryptography and Network Security - 19th International Conference ACNS Part II, volume 12727 of *Lecture Notes in Computer Science*, Springer, Kamakura (Japan), 2021, pp. 424–447. URL: https://doi.org/10.1007/978-3-030-78375-4_17. doi:DOI 10.1007/978-3-030-78375-4_17.

[31] Trusted Computing Group, Trusted Platform Module 2.0 Library Part 1: Architecture, TCG Review, 2025. https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-2.0-Library-Part-1_Architecture-V185-RC2_10July2025.pdf.

[32] The TORSEC Research Group, Post-Quantum Firmware Trusted Platform Module Implementation, 2026. https://github.com/torsec/ms-tpm-20-ref/tree/pq-ftpm.

[33] IMA Documentation, 2025. https://ima-doc.readthedocs.io/en/latest/index.html.

[34] TCG, Trusted Platform Module 2.0: A Brief Introduction, 2019. https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf.

[35] NIST, Transition to Post-Quantum Cryptography Standards, 2024. doi:DOI 10.6028/NIST.IR.8547.ipd.

[36] I. G. T. www.globalplatform.org | © 2011-2022 GlobalPlatform, TEE System Architecture v1.3, Internet-Draft, Global Platform, 2022. URL: https://globalplatform.org/wp-content/uploads/2022/

05/GPD_SPE_009-GPD_TEE_SystemArchitecture_v1.3_PublicRelease_signed.pdf.
[37] TCG, TCG TSS 2.0 Overview and Common Structures Specification, 2021. https://trustedcomputinggroup.org/wp-content/uploads/TSS_Overview_Common_v1_r10_pub09232021.pdf.
[38] TCG, TCG Feature API (FAPI) Specification, 2020. https://trustedcomputinggroup.org/wp-content/uploads/TSS_FAPI_v0p94_r09_pub.pdf.
[39] TCG, TCG TSS 2.0 Enhanced System API (ESAPI) Specification, 2021. https://trustedcomputinggroup.org/wp-content/uploads/TSS_ESAPI_v1p0_r14_pub10012021.pdf.
[40] TCG, TCG TSS 2.0 System Level API (SAPI) Specification, 2021. https://trustedcomputinggroup.org/wp-content/uploads/TSS_SAPI_v1p1_r36_pub10012021.pdf.
[41] TCG, TCG TSS 2.0 Marshaling/Unmarshaling API Specification, 2020. https://trustedcomputinggroup.org/wp-content/uploads/TCG_TSS_Marshaling_Unmarshaling_API_v1p0_r07_pub.pdf.
[42] TCG, TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification, 2020. https://trustedcomputinggroup.org/wp-content/uploads/TCG_TSS_TCTI_v1p0_r18_pub.pdf.
[43] TCG, TCG TSS 2.0 TAB and Resource Manager Specification, 2019. https://trustedcomputinggroup.org/wp-content/uploads/TSS_2p0_TAB_ResourceManager_v1p0_r18_04082019_pub.pdf.
[44] Linux TPM2 & TSS2 Software , OSS implementation of the TCG TPM2 Software Stack (TSS2), 2026. https://github.com/tpm2-software/tpm2-tss.
[45] Parsec, TSS 2.0 Rust Wrapper over Enhanced System API, 2026. https://github.com/parallaxsecond/rust-tss-esapi.
[46] Parsec, The Parsec Book, 2026. https://parallaxsecond.github.io/parsec-book/.
[47] OP-TEE Keylime deployment, 2026. https://github.com/torsec/manifest.