POLITECNICO DI TORINO
Repository ISTITUZIONALE

Implicit end-point attestation for trusted channel establishment in the keystone TEE

Original
Implicit end-point attestation for trusted channel establishment in the keystone TEE / Bruno, G., Sisinni, S., Bravi, E., Ferro, L., Ciravegna, F., Lioy, A.. - In: COMPUTER NETWORKS. - ISSN 1389-1286. - 288:(2026). [10.1016/j.comnet.2026.112603]

Availability:
This version is available at: 11583/3013770 since: 2026-07-30T16:34:39Z

Publisher:
Elsevier

Published
DOI:10.1016/j.comnet.2026.112603

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

19 November 2026

Computer Networks 288 (2026) 112603

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Implicit end-point attestation for trusted channel establishment in the keystone TEE☆,☆☆

Giacomo Bruno, Silvia Sisinni*, Enrico Bravi, Lorenzo Ferro, Flavio Ciravegna, Antonio Lioy

Politecnico di Torino - Dipartimento di Automatica e Informatica, Corso Duca degli Abruzzi 24, Torino, 10129, Italy

ARTICLE INFO

Keywords:
Trusted computing
DICE
Implicit attestation
Trusted execution environment
Remote attestation

ABSTRACT

Nowadays, various contexts and applications include IoT devices. Due to their limited resources and power constraints, these systems are a possible threat vector that adversaries may exploit for cyberattacks. Despite the protection of network communication with Secure Channels, such as TLS and IPsec, the endpoint with which we exchange information may still be compromised. Thus, an adversary may obtain access to sensitive information or send corrupted data without being detected by the other network nodes. Remote Attestation is a possible security control that can detect device misbehaviours, allowing an external entity to verify a platform's trustworthiness. Attestation reports contain system measures and configuration information to prove the system state. So, the external entity can verify the platform's authenticity and integrity by comparing the data included in the report with the corresponding expected values. Several solutions addressing this issue are presented in the literature, including Remote Attestation over secure channels. Trusted Channels is the name given to these protocols, as the trustworthiness of the endpoints is a security property guaranteed by the Channel, in addition to the properties of a secure channel. This paper proposes a new certification protocol for IoT devices that merges Remote Attestation with the issuance of a certificate for a key pair generated and stored securely on the platform. This new credential enables the establishment of TLS channels; therefore, the other endpoint obtains information about the node's trustworthiness. We implemented the protocol within the Keystone framework, which enables a customisable Trusted Execution Environment that provides the Remote Attestation mechanism.

1. Introduction

Today, the widespread adoption of remote services and network connections has necessitated a higher level of security in communications between network nodes. This paper focuses on the Internet of Things (IoT) [1] and embedded devices, as they are adopted in different contexts, such as healthcare, transportation, and manufacturing. These devices are even more in need of strong security protections [2] since they are the target of several categories of attacks [3]. This issue is due to the limited resources and computational capabilities of IoT platforms. In some cases, the possibility of an attacker physically accessing a device extends the attack surface. For this reason, the adoption of Secure Channels, such as Transport Layer Security (TLS) [4] or Internet Protocol Security (IPsec) [5], to protect the communication between network nodes may become insufficient. If an adversary compromises one of the endpoints, they can access the received sensitive information or send corrupted data. While it is not always possible to protect a device from all types of attacks, it is essential to be able to detect its compromise promptly. This process utilises Remote Attestation mechanisms [6–8], enabling external third parties to verify the platform's authenticity and integrity. To guarantee these properties, the node to be attested generates reports that include system measures and configuration information. Then, the external entity compares the received values with the reference ones, and if they match, the node is

☆ This article is part of a Special issue entitled: 'Cybersec Attacks and Defenses' published in Computer Networks.

☆☆ This work has received funding from the SPIRS (Secure Platform for ICT Systems Rooted at the Silicon Manufacturing Process) project with Grant Agreement No. 952622 under the European Union's Horizon 2020 research and innovation programme. This work was also partially supported by the project SERICS (PE00000014) under the NRRP MUR program, funded by the European Union - NextGenerationEU. This publication is also part of the project PNRR-NGEU which has received funding from the MUR – DM 352/2022.

* Corresponding author.
E-mail addresses: giacomo.bruno@polito.it (G. Bruno), silvia.sisnni@polito.it (S. Sisinni), enrico.bravi@polito.it (E. Bravi), lorenzo.ferro@polito.it (L. Ferro), flavio.ciravegna@polito.it (F. Ciravegna), antonio.lioy@polito.it (A. Lioy).

https://doi.org/10.1016/j.comnet.2026.112603
Received 27 October 2025; Received in revised form 29 June 2026; Accepted 19 July 2026
Available online 28 July 2026
1389-1286/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

trusted. Several designs have been published in the literature to merge Remote Attestation with Secure Channel protocols. These new protected communications are called Trusted Channels since the endpoint trustworthiness is added to the other security properties the Secure Channel offers (i.e., integrity, confidentiality, traffic authentication, and endpoint authentication). Thanks to this new security control, a possible attack on the network node can be detected. This way, the other endpoint can interrupt the connection setup or tear down the established channel. Among the technologies that provide the Remote Attestation mechanism, we adopted the Trusted Execution Environment (TEE) [9], a processing environment that allows application execution to be isolated from the rest of the system. Thus, the endpoint is protected from attacks generated from the untrusted part of the host.

This paper proposes the issuance of credentials, bound to the endpoint's integrity status, to be used in the secure channel. The associated key pair is computed starting from system measures and configuration to achieve it. For this purpose, we adopt the TCG Device Identifier Composition Engine (DICE) specification [10] to bind the values of the keys to the subject's identity, following our previous work [11–13]. Then, a certification protocol merges the Remote Attestation of the endpoint to the issuance of X.509 certificates for those keys. This process ensures that only a specific application running on a given platform can generate the key pair included in the credential and use it in authentication during the channel establishment. The solution is implemented in Keystone [14], a framework that allows the development of customisable TEEs on RISC-V hardware. Thanks to this project, developers are not obliged to adopt a specific chip and can adapt the TEE to their needs.

The main difference with respect to previous trusted channel designs is that the attestation evidence is not exchanged during the establishment of each channel. In our solution, Remote Attestation is performed before certificate issuance. The certificate is issued only for an identity key generated from the measured state of the trusted application and protected by the TEE. Therefore, when this key is later used in the TLS handshake, its usage implicitly proves that the endpoint corresponds to the attested application state. The remote peer only verifies a standard certificate chain and does not need to know the attestation format, the DICE certificate chain, or the specific TEE adopted by the platform.

Paper Structure. The paper is organised as follows. Sections 2 and 3 provide an introduction to trusted channels based on the solutions presented over the years, and go deeper into the protocols more related to our proposal. Section 4 describes the design of the certification protocol and the procedure followed to generate the key pair, the security of which is evaluated in Section 5. Section 6 presents a possible protocol implementation, describing the several entities involved in the certification process. We tested the performances of those applications, reporting the main results in Section 7. Finally, Section 8 summarises the entire work, presenting also the possible future works.

2. Background

2.1. Trusted computing

Trusted Computing is a security paradigm developed over the last decades to ensure the correct execution and behaviour of an information system. The Trusted Computing Group (TCG) [15] is one of the main contributors defining standards for this technology. A crucial concept defined by the TCG is the Trusted Platform (TP). A TP is, in general, a system which is able to produce a proof of its internal state and provide it to an external relying party. To build a TP, different Root of Trusts (RoTs) are necessary. A RoT is a platform component which is considered *trusted* by design, and whose malfunction cannot be detected at runtime. The TCG defined the three minimal RoTs to build a TP:

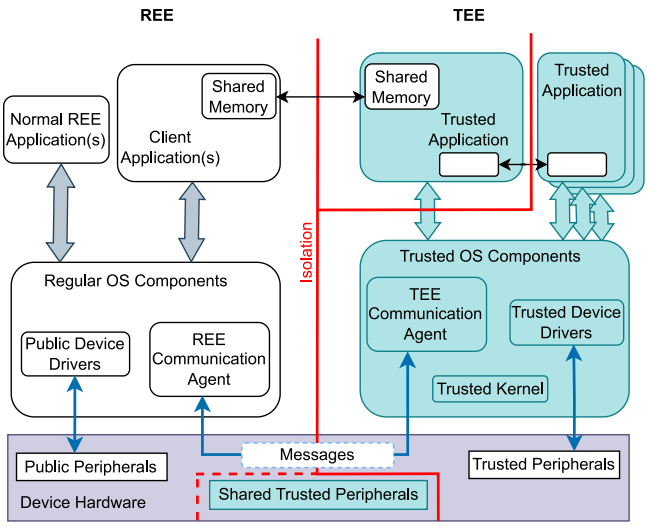


Fig. 1. TEE isolation architecture [21].

- *Root of Trust for Measurement (RTM)*: it is the first software or hardware component of a system, usually called Core Root of Trust for Measurement (CRTM), which takes control of the platform, and it has the purpose of producing a measurement (e.g., digest) of the following components that will be executed;
- *Root of Trust for Storage (RTS)*: it is the component in charge of providing secure (e.g., tamper-proof) storage where the measurements produced by the RTM can be stored;
- *Root of Trust for Reporting (RTR)*: it is in charge of reporting the measurements stored in the RTS in order to prove its internal state to external third parties.

Among the solutions that have been published, the Trusted Platform Module (TPM) [16] specification by the TCG, typically implemented as an external chip, provides many security primitives. In particular, it acts as an RTS and RTR.

To externally verify the trustworthiness of a platform, Remote Attestation (RA) techniques can be used [17]. RA is typically a challenge-response procedure, where a trusted entity (*Verifier*) retrieves and verifies the internal state of a remote platform (*Attester*). RA techniques can vary based on the assumptions and threat model defined for the specific scenario. These approaches can be divided in three main categories:

1. *Hardware-Based RA*: these techniques exploit a platform's hardware capabilities to ensure strong security properties. They are based on the use of a hardware RoT, such as the TPM, DICE, and Intel SGX [18].
2. *Software-Based RA*: these techniques have weaker security assumptions than the hardware-based ones. They are based solely on software components [6,7], making them suitable for scenarios where hardware components cannot be utilised.
3. *Hybrid RA*: these techniques [19,20] combine hardware-based and software-based approaches, in order to obtain the most suitable requirements for the specific scenario. In this case, the security assumptions are on hardware and software components, introducing more flexibility.

2.2. Trusted execution environment

A TEE aims to execute critical applications, or eventually only critical operations, in a shielded and isolated environment. They typically leverage hardware security extensions that aim to provide isolation of

memory and execution context at the hardware level. Several technologies exist, each with a distinct design and security assumptions. For this reason, the Global Platform Association proposed a general specification [21] to provide a common base for building and developing new TEEs, aiming to establish a standard architecture for these technologies.

The common architecture of a TEE (Fig. 1) is composed of two execution environments: the Rich Execution Environment (REE) and the TEE. The REE is the "normal" execution environment, where there is no hardware-enforced isolation mechanism and the instructions are stored and executed without protection. The TEE is instead a hardware-protected and isolated environment, where Trusted Applications (TAs) run isolated from the REE and can also run isolated from other TAs.

The interactions between the REE and the TEE are managed at the firmware level, where a Secure Monitor is responsible for performing the context switch and configuring the hardware to ensure protection. The two different execution environments then have completely different software stacks. They have their own kernel, with their own drivers and applications.

Some TEE technologies are defined as two different environments, the REE and the TEE, without providing more isolation, such as among different TAs. One example is the ARM TrustZone [22] technology. In this case, the device is divided into Normal World and Secure World, where the Secure World is hardware-protected from the Normal World; however, no isolation or protection is provided among the TAs deployed.

In contrast, other technologies, such as Intel SGX [18] or Keystone [14], offer a more granular architecture. In this case, isolation is achieved at the TA level for Keystone and at the code level for Intel SGX.

2.3. Trusted channels

In recent years, several research groups have proposed designs for trusted channels, each with its pros and cons. Initially, the integrity and trustworthiness of the platform are ensured by employing a TPM [16]. This component enables the monitoring of the platform's behaviour and allows trusted external parties to verify whether the current status aligns with the reference status, thereby identifying any anomalies in the system. If the check is successful, the platform is considered trustworthy, and an attestation result is generated, allowing other parties to verify its validity without needing a reference value.

The information required by this protocol, known as Remote Attestation, is conveyed through an attestation report. It contains the measurements of the system, e.g., hash of the memory where the software binaries are stored, together with other information about the platform, signed by an asymmetric key pair owned and managed securely by the TPM. However, in more recent works, researchers have considered another technology for designing trusted channels, namely TEE [9]. Compared to the TPM, this new framework enables the execution of applications in a secure environment, isolated from the usual unprotected system, known as REE. Among the several products available, some of the TEEs considered in the designs are OP-TEE (based on ARM TrustZone [22] technology) and Intel SGX [18].

Despite the adoption of trusted computing technology, relay and collusion attacks remain two of the primary threats that must be considered when designing the trusted channel protocol. Both are based on the unsecured linkage between the Remote Attestation session and the secure channel. Therefore, it is possible that an adversary attempting to establish a trusted channel may attest to a trusted platform and then forward the received attestation reports, impersonating that system or at least its trustworthiness. For this reason, a simple Remote Attestation session inside a secure channel, e.g., TLS, is not enough, making the design and development of the trusted channel more challenging.

The trusted channels proposed in the literature can be classified into three main groups, depending on the approach followed to implement them:

1. new custom protocols (e.g., [23]), that propose ways to perform a handshake in which the keys of the secure channel (session key, authentication key) are bound to the attestation session and key;
2. modifications to the standard secure channel protocols (e.g., [24]), for instance, how the TLS session key is computed by integrating the material obtained through the Remote Attestation;
3. new extensions in secure channels and X.509 [25] certificates (e.g., [26]), to convey the attestation reports without requiring any further modification to the current implementations and libraries for the secure channel.

Among these solutions, those that propose a customised trusted channel solution offer greater freedom in design choices, with the possibility of integrating Remote Attestation in various ways during the handshake. However, one of the disadvantages of these solutions is the need to implement them from scratch or at least modify existing libraries. This would not only increase implementation time and cost but would also result in a lack of backward compatibility with legacy or obsolete systems. Therefore, integrating Remote Attestation only through extensions to the secure channel and the used certificates could result in faster and more widespread adoption.

Another problem related to trusted channels is the usage of standards for Remote Attestation. The issue arises because a node must be able to interpret the format of attestation proofs to assess the trustworthiness of a remote peer. This limits the establishment of trusted connections to only those systems that follow the same specifications, making the channel dependent on the attestation framework used by the platform. Working groups such as the IETF's Remote ATtestation ProcedureS (RATS) [27] and the TCG [15] are developing standards for Remote Attestation that enable interoperability between different solutions, making them easier to use and integrate. Among these documents, there are proposals of standard extensions used to convey attestation reports and results in TLS [28] or X.509 data structures [29].

Trusted channels can also be classified based on the network layer where they operate. Most of the proposed solutions are built on top of the transport layer, starting from TLS. However, more recent designs [30,31] developed a trusted channel on top of the HTTP protocol. An advantage of this solution is the exchange of attestation information after the secure channel is set up. In fact, in the other protocols, the platform measurements and configuration are exchanged during the secure channel handshake, with the possibility of leaking that data to untrusted or unidentified hosts. On the contrary, the exchange of attestation data after establishing the secure channel guarantees at least that the other node has been authenticated previously. However, the disadvantages of building a trusted channel at the application layer are the postponed check about entity trustworthiness, which can increase the attack surface, and the solution's limited applicability to a specific context, such as HTTP web traffic.

The last problem considered during the development of trusted channels is the certification cost. In this case, two certificate chains are required: one refers to the credential used to authenticate the peer in the secure channel, and the other to the key pair used to sign the attestation reports. Moreover, if the second one depends on the measurement and configuration of components, every system update results in newly generated keys and certificates.

3. Related work

Intel researchers, in a paper published in 2019 [32], proposed integrating Intel Software Guard Extensions (SGX) attestation into TLS. Two aspects of this design are relevant for the comparison with our proposal. The first one is the binding between the TLS authentication key and the enclave instance. To obtain this property, the enclave generates a fresh key pair at each startup and includes the hash of the public key in the attestation report. In this way, the verifier can check

that the key used during the TLS handshake belongs to the attested enclave. The second aspect is the position of the attestation exchange in the protocol. The attestation proof is provided to the remote endpoint during connection setup, so that the peer can evaluate the enclave state before accepting the channel. This is achieved by inserting the generated proof into the X.509 certificates as a custom extension. This way, it can be checked with simple hooks for certificate verification.

The solution proposed by Intel supports two SGX attestation models. The Enhanced Privacy ID (EPID) protocol protects the identity of the attested subject by using group keys. In this case, the relying party leverages the Intel Attestation Service to assess the received evidence. The other model is ECDSA attestation, which enables third parties to verify reports using non-Intel attestation infrastructure in a standard manner. The use of certificates for periodically regenerated keys is one of the main drawbacks of this solution. However, the fact that they are self-signed significantly reduces the certification cost. Moreover, legacy clients may reject these credentials and abort the connection in the case of unrecognised custom extensions. A possible solution to this problem is using an intermediary CA to issue the credential with a protocol such as Automatic Certificate Management Environment (ACME) [33]. Additionally, timestamps to ensure the freshness of attestation reports remain an issue, as the verifier must be able to access a trusted time source. This approach shows that enclave attestation can be integrated with TLS without redesigning the whole secure channel. Its main outcome is the binding between the TLS authentication key and a specific enclave instance. However, the attestation proof is still transported during the connection setup, and the peer has to understand and appraise the SGX evidence. The solution is therefore strongly related to the SGX attestation model and to the verification logic required by the remote endpoint.

RATLS [34] follows a similar direction and integrates Remote Attestation into TLS through a companion library for OpenSSL [35,36]. The main result of this work is a practical integration of attestation in an existing TLS stack, without requiring the definition of a new secure channel protocol. The attestation exchange and the TLS handshake are cryptographically linked, so the channel terminates in the attested environment. However, attestation evidence is still processed during the establishment of the channel. Thus, the peer remains involved in the verification of attestation-specific data.

Another recent work [37] proposes a trusted channel protocol for confidential workloads based on WireGuard [38] and integrated with the Confidential Containers project [39]. This work shows that trusted channels are useful also outside the TLS scenario, especially for confidential computing deployments. Its main outcome is the establishment of a protected channel towards an attested workload, while keeping compatibility with unmodified client applications. However, the attestation phase is still part of the initial channel establishment and the solution is tied to a specific deployment stack.

An active IETF draft [28] currently defines a set of extensions to the TLS Handshake. They carry attestation reports and results to demonstrate endpoint trustworthiness supporting the two attestation models defined by RATS [40]. This way, the Remote Attestation session is tied to the TLS authentication key. These drafts address another limitation of previous works, i.e., the lack of common formats for carrying attestation evidence and attestation results. Their main outcome is the definition of standard protocol elements that can improve interoperability between different implementations. At the same time, the endpoint remains attestation-aware, since it still has to receive, parse, and verify attestation-related data during the protocol execution. Other extensions are defined in a draft for ACME [41]. It aims to integrate device attestation into the automatic certificate issuance process, ensuring the subject's trustworthiness before generating the credential. The ACME extension moves part of the check to the certificate issuance process, but its goal is certificate management for attested devices. It does not define the DICE-based generation and use of a TEE-protected endpoint identity key adopted in this work.

Table 1
Positioning of the proposed solution.

Approach	Attestation check	Evidence to peer	Channel impact
Custom or non-TLS trusted channels	Channel setup	Yes	New or adapted channel logic
TLS modifications	TLS handshake	Yes	Modified TLS logic
TLS/X.509 extensions	TLS handshake	Yes	Extensions and verification hooks
This work	**Certificate issuance**	**No**	**Standard TLS after certification**

The publications described so far show the interest of various groups in integrating RA in secure channel protocols. Our solution follows the same objective, but differs in the point where the attestation result is used. In previous approaches, the attestation evidence is usually bound to the channel establishment phase, by modifying the secure channel protocol or by carrying evidence in TLS or X.509 extensions. In our proposal, the attestation check is instead used before the channel is created, during the issuance of the endpoint credential. The following TLS handshake remains standard, and the remote peer verifies only the certificate and the signature produced with the certified key. This reduces the amount of attestation-specific information exchanged during channel establishment and avoids requiring each peer to appraise platform-specific evidence.

The main design choices are summarised in Table 1. The table does not report implementation details, but only shows where the attestation check is performed, whether the remote peer receives platform evidence, and how much the secure channel protocol is affected.

Differently from the approaches reported above, our proposal uses the attestation result during certificate issuance. After this step, the protected channel is established as a standard TLS channel. The remote peer does not receive attestation evidence, but only verifies the certificate and the signature produced with the certified key. Since this key is bound to the measured endpoint state and protected by the TEE, its use implicitly proves the endpoint trustworthiness.

4. Protocol design

4.1. Threat model

In the threat model considered for the design of the solution, the adversary has complete control over the network, i.e., he can read, modify, or drop network packets. The main attacks considered are the Man in the Middle (MITM) and relay attacks, where the adversary compromises the authentication and Remote Attestation steps, exploiting an incorrect binding between the evidence and the channel. Moreover, the main threats against cryptographic primitives, such as cryptanalysis, are considered.

From a system security perspective, the adversary can control the untrusted software stack of the endpoint, including the host application and the REE. For this reason, trusted channel endpoints are critical applications that need to be protected by the TEE and run in secure partitions called "enclaves", which are isolated from the REE. The adversary can invoke the interfaces exposed to the untrusted world, but cannot directly read or modify enclave memory if the TEE Trusted Computing Base (TCB) behaves correctly.

The TCB of our implementation is the one required by Keystone for enclave isolation and attestation. It includes the hardware RoT, the RISC-V Physical Memory Protection (PMP) mechanism used for memory isolation, and the Security Monitor (SM) functions that create enclaves and manage their life-cycle, compute measurements, and operate with DICE keys. We assume that these components correctly implement the operations required by the protocol. This assumption follows the Keystone threat model [14], and recent work has started to

formally verify part of this basis, such as the RISC-V PMP mechanism used for memory isolation [42]. However, the complete verification of a TEE TCB is outside the scope of this paper. This assumption does not mean that these components are proved correct in this paper, but defines the boundary on which the protocol is built. Therefore, implementation vulnerabilities in the SM, in the hardware isolation mechanism, or in the DICE key derivation have to be considered as a compromise of the TEE TCB. In that case, the certification protocol cannot preserve the binding between the measured enclave and the certified key. This limitation is common to protocols built on top of a TEE: the protocol can use the isolation and measurement guarantees provided by the platform, but it cannot repair a compromised TCB.

Physical attacks and side-channel attacks are not considered in this work. This includes attacks that extract secrets from the chip or from external memory, fault-injection attacks, cache or timing attacks, and speculative-execution attacks. These threats require platform-specific hardware or implementation countermeasures and are usually studied separately from the secure-channel protocol. The proposed certification protocol can be used with a TEE that provides stronger protection against these attacks, but it does not introduce such protection by itself.

4.2. Requirements

The main purpose of the solution we propose in this paper is to ensure the same security properties as a TLS channel, ensuring that the endpoint behaves as expected. In this solution, two aspects ensure the trustworthiness of nodes: (1) the generation of the keys used in the TLS authentication has to be bound to the integrity state of the components running in the platform, so a modification of them must result in a different key pair; (2) a new certification protocol between the trusted application and an external certification authority that issues the certificate after attesting the requester.

Starting from the solutions in the literature (e.g., [26]), the security requirements (SR) are:

1. *Secure channel properties*: the new protocol has to provide the same security properties of a standard secure channel, i.e., integrity and confidentiality of messages, authentication of data and nodes, and freshness to prevent replay attacks;
2. *Authentic linkage of configuration/integrity information and secure channel*: the Remote Attestation session and the secure channel have to be bound securely during the handshake to prevent relay attacks;
3. *Privacy* (least information paradigm): the trusted channel requires only the essential information about the platform needed to ensure the integrity and trustworthiness of the endpoint.

Among the above security requirements, we have focused more on the SR3, as the proposed trusted channel protocol aims to reduce the possibility for adversaries to obtain unwanted information about the system, or at least to avoid profiling it. Then, the following functional requirements (FR) complete the list of properties that the solution has to satisfy:

1. *Fast deployment support at the application level*: the trusted channel should be based on standards and existing software environments, requiring minimal modifications to applications and libraries;
2. *Minimal handshake overhead*: the performance of the trusted channel should be comparable with the secure channel;
3. *Flexible configuration/integrity reporting*: the solution should be TEE-agnostic so that endpoints can establish trusted channels independently from the attestation technology adopted;
4. *Backward compatibility*: the endpoints should establish secure channels towards systems that do not support the trusted channel protocol.

4.3. Protocol assumptions and design challenges

The protocol is built on a small set of operational assumptions. The external certification authority and the Verifier are trusted entities, and the Verifier has access to the reference measurements (golden values) of the components whose trustworthiness has to be verified. The certificates of trustworthy platform manufacturers or owners are configured as trust anchors, so that the DICE certificate chain can be verified as being based on a trust anchor before issuing a new credential. Finally, the TEE TCB provides enclave isolation, correct measurements, and protected use of the keys, as stated in the threat model (Section 4.1).

Starting from these assumptions, the protocol has to solve three main problems. The first one is the binding between the Remote Attestation session and the key used later in TLS, because a weak binding would enable relay attacks. The second one is privacy: measurements and configuration values should be visible only to the entities that need them for certification. The third one is deployability, since the final channel should remain a standard TLS channel and should not require the peer to parse attestation evidence. These challenges motivate the certification protocol described below, where the key used for TLS authentication is generated and used through the SM.

4.4. Proposed solution

As described in the previous section, the protocol we defined is based on the assurance that certificates are generated only for trustworthy applications. The essential element of the protocol is the key pair used to establish the TLS channel, so its generation must follow specific security properties. Firstly, each key pair must be cryptographically bound to the software integrity of a specific enclave running the TEE. To be achieved, these keys are generated at every reboot based on system measurements and configuration. As a result, any modification to the trusted application or its dependencies will result in a different key pair, making previously issued certificates useless. Then, the attack surface available to the adversary reduces by delegating the management of the key to the SM. The SM is a component of the TEE that acts as a separation kernel between the trusted and untrusted worlds and also as a trusted kernel of the TEE. So, the security of the private key depends on the security of the adopted TEE and its components. Due to this design choice, the trusted application must invoke specific functions that the SM offers to operate with those key pairs. Once generated, the corresponding certificates are issued, binding them to the subject's identity. However, unlike the well-known Certificate Signing Request (CSR), this solution includes attestation proof of the subject to demonstrate its trustworthiness before issuing the credential. Therefore, when this certificate is used to establish a secure channel, the other endpoint has assurance that the authentication key belongs to a trustworthy node. This mechanism is the basis of the implicit endpoint attestation provided by the proposed trusted channel.

4.5. Hardware RoT integration

The TCG DICE Layering Architecture [10] defines the properties of the key pair. This specification describes an architecture that implements security and privacy technologies with minimal silicon requirements, instrumental in contexts such as IoT devices. For example, it is possible to use simple silicon capabilities and software techniques to define a cryptographically strong device identity. Features such as attestation and software updates require a device with this capability. The architecture is based on the computation of identifiers for the components of the TCB, the set of hardware, firmware and software components critical to its security. These identifiers are called Compound Device Identifiers (CDIs) and are calculated as shown in Fig. 2. Each level in the boot chain calculates the CDI for the following (ith) layer using a One-Way Function (OWF), which uses the CDI of the

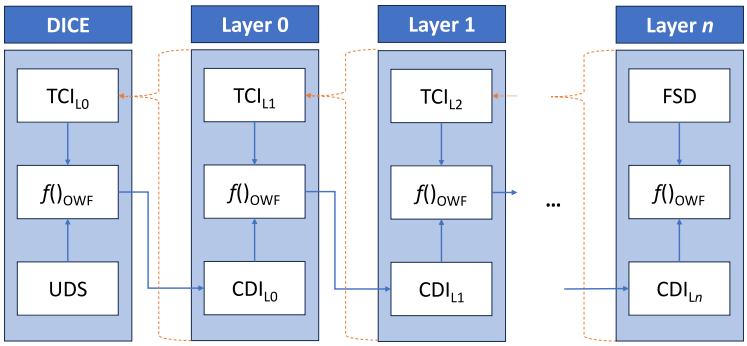


Fig. 2. DICE TCB Layering Architecture [10].

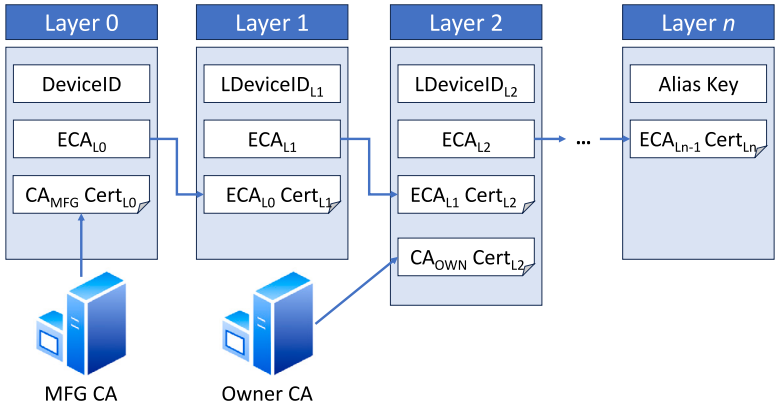


Fig. 3. DICE embedded certificates mechanism.

current ($(i-1)$th) layer and the measurement of the ith component, called TCB Component Identifier (TCI). The value thus calculated is then provided to layer ith to continue the chain. This process starts in the DICE Core, which is assumed to be trusted since its misbehaviour cannot be detected. To compute layer 0's CDI, DICE Hardware Root of Trust (HRoT) uses the Unique Device Secret (UDS), a statically unique and device-specific secret value generated at the time of device manufacture or provisioning. Due to the way CDIs are generated, if one of the components in the chain is changed or replaced, its TCI will change and consequently all CDIs from that layer on.

The DICE Layering Architecture also defines how asymmetric and symmetric keys can be derived from the CDI of the layer they refer to. In addition, the specification describes how these keys can be certified (Fig. 3). Components can use a standard Certificate Authority (CA), named eXternal Certificate Authority (XCA) or a CA inside the device and managed by a DICE layer, which acts as Embedded Certificate Authority (ECA). For these reasons, protecting the identifiers is imperative to prevent adversaries from calculating the keys' value. Among the types of asymmetric keys considered in the document [10], the three essential keys used in the proposed trusted channel are: (1) ECA Key, which signs certificates for the current or subsequent TCB layer; (2) Local Attestation Key (LAK), used to sign attestation evidence; (3) Local Device Identifier (LDevID) Key, an identity key that signs authentication challenges, e.g., during the TLS handshake. These keys are derived deterministically from the CDIs and TCIs of the TCB components. For this reason, signing with them implicitly states the Layered Identity. Thus, by checking the signature validity and the signer's identity, the Verifier can attest to the platform's trustworthiness since only a specific configuration of a protected application could have generated that key.

Moreover, the specifications of the DICE define how to build a chain of ECAs to issue certificates for the generated keys. Different from the standard credential issuance, these processes include the attestation of the requester. When an ECA issues the certificate, the measures of the requester are computed directly from the component that acts as ECA. Otherwise, if an XCA generates the certificate, the trustworthiness of the subject and the underlying components is remotely attested before generating the new credential. This way, if the external party relies on the implementation of DICE, TCB, and XCA, trusting in the security properties they guarantee, the usage of the key implies the correct configuration of the system at load time.

The threats in this context include cryptanalysis, which allows the attacker to deduce the private key starting from the output of asymmetric key operations. Thus, the proposed protocol defines two different keys for each enclave, an LAK by which the Attester signs the attestation evidence for that specific trusted application, and an LDevID Key, which is the endpoint for authentication during the secure channel handshake.

4.6. Formalisation and binding properties of the certification protocol

The relation between measurements, keys, evidence, and certificates can be described by modelling the values that are bound during certificate request and during the following TLS authentication. Let TA be the trusted application requesting the certificate, and let ID_{TA} be its identity. For each component X, TCI_X denotes the measurement of its code and configuration, while $Ref(X)$ denotes the corresponding reference value. Let D_{TA} be the set of components on which TA depends and whose measurements are included in the DICE chain. In our implementation, this set includes the SM and the hardware RoT. Finally, let F be the OWF used in the DICE chain.

Starting from the device secret, the identifiers of the layers are computed as follows:

$$CDI_0 = F(UDS,\ TCI_0), \qquad CDI_i = F(CDI_{i-1},\ TCI_i)$$

The identifier associated with the trusted application layer, denoted as CDI_{TA}, is then used by the SM to derive two different key pairs:

$$(LDevID.PK_{TA},\ LDevID.SK_{TA}) = \text{Derive}(CDI_{TA},\ ''LDevID'')$$

and

$$(LAK.PK_{TA},\ LAK.SK_{TA}) = \text{Derive}(CDI_{TA},\ ''LAK'')$$

The LDevID key is used for endpoint authentication, while the LAK key is used to sign the attestation evidence.

When the XCA sends a fresh *nonce*, the SM builds the attestation message:

$$m_{att} = TCI_{TA} \parallel LDevID.PK_{TA} \parallel nonce$$

The SM signs this value with the LAK private key:

$$E_{att} = \text{sign}(m_{att},\ LAK.SK_{TA})$$

This evidence binds the measured state of the trusted application (TCI_{TA}), the public key for which the certificate is requested ($LDevID.PK_{TA}$), and the freshness value generated by the XCA (*nonce*).

The CSR contains the identity of the requester, the LDevID public key, the nonce, the attestation evidence, and the DICE certificate chain. The data to be signed in the CSR are:

$$m_{CSR} = ID_{TA} \parallel LDevID.PK_{TA} \parallel nonce \parallel E_{att}$$

The CSR signature is computed with the corresponding LDevID private key:

$$\sigma_{CSR} = \text{sign}(m_{CSR},\ LDevID.SK_{TA})$$

Thus, the request can be written as:

$$CSR_{TA} = (ID_{TA},\ LDevID.PK_{TA},\ nonce,\ E_{att},\ ChainDICE_{TA},\ \sigma_{CSR})$$

The XCA and the Verifier accept the request only if all the following conditions hold:

$$\begin{aligned} Accept(CSR_{TA}) = 1 \iff\ & \text{ValidChain}(ChainDICE_{TA}) \ \wedge\ \text{Fresh}(nonce) \ \wedge \\ & \left(\forall X \in D_{TA} \cup \{TA\} : TCI_X = Ref(X)\right) \ \wedge \\ & \text{Verify}(LAK.PK_{TA},\ m_{att},\ E_{att}) = 1 \ \wedge \\ & \text{Verify}(LDevID.PK_{TA},\ m_{CSR},\ \sigma_{\text{CSR}}) = 1 \end{aligned}$$

The first condition verifies that the DICE certificate chain is rooted in a trusted anchor. The measurement check verifies that the trusted application and its dependencies correspond to the expected reference values. The verification of E_{att} checks that the attestation evidence was generated by the LAK associated with the trusted application. The verification of σ_{CSR} checks that the requester also controls the LDevID private key corresponding to the public key included in the CSR.

If these checks are successful, the XCA issues the certificate for the LDevID public key:

$$Cert_{TA} = \text{Cert}_{\text{XCA}}(ID_{TA},\ LDevID.PK_{TA})$$

During the following TLS handshake, the trusted application authenticates itself by signing the TLS authentication data, denoted as $data_{TLS}$:

$$\sigma_{TLS} = \text{sign}(data_{TLS},\ LDevID.SK_{TA})$$

The remote peer verifies the certificate issued by the XCA and the TLS signature. Therefore, the same key that was certified after attestation is the key used during channel establishment.

The model shows two bindings. The first one is the binding between the attestation evidence and the certificate request, because the evidence signs the trusted application measurement, the LDevID public key, and the nonce from the XCA. The second one is the binding between the certificate and the TLS channel, because the TLS authentication is performed with the private key corresponding to the LDevID public key, certified after endpoint attestation. Under the assumptions defined in Section 4.1, this provides the implicit endpoint attestation property: a successful TLS authentication with $Cert_{TA}$ proves that the endpoint uses the identity key certified after verification of the trusted application state.

4.7. Certification protocol

DICE ECAs build a chain of certificates from a trusted root (e.g., the device manufacturer certificate) up to the trusted application credential. This chain includes the certificate of the DICE HRoT's key pair, an ECA key signed by the manufacturer and embedded in the device before deployment. Thanks to these credentials, it is possible to ensure the identity and trustworthiness of the layers from the HRoT to the trusted application. For this purpose, a new certificate extension is included in the credential to convey the subject's measurements and configuration. However, a malicious host may intercept that information, with the possibility of leaking sensitive data that is usable to prepare attacks against the platform or only profiling it, becoming a risk for privacy. For this reason, the solution in this paper presents a certification protocol for a LDevID that involves an XCA. As introduced before, this solution primarily aims to issue certificates only for trustworthy applications, allowing other parties to rely on the XCA verification. Furthermore, since privacy is considered, introducing the XCA generates another certificate chain that hides the DICE one and can eventually use aliases for original applications. According to RATS Architecture [40], this solution adopts the Background Check topology model. Here, the Attester (i.e., the TA, the entity under verification), provides this data directly to the Relying Party (i.e., the XCA). Then the Verifier (Ver) receives the proofs and evaluates them against the appropriate reference values, returning the outcome as a signed attestation result message.

Section 4.6 formalises the binding between measurements, keys, evidence, and the final certificate. The protocol in Fig. 4 describes how this binding is obtained through the messages exchanged by the TA, the SM, the XCA, and the Verifier. Initially (1.1/1.3), the TA generates the LDevID key pair by invoking a function of SM's interface. In this way, the key pair is computed and stored securely inside the SM, avoiding its exposure to the TA, which would break the binding between the component and the value of the key. Inside the SM's function, the LDevID is certified using a DICE ECA to demonstrate the generation of the key and its identity according to DICE specifications. The SM returns to the TA only the LDevID public key as a key identifier and its certificate.

In the second phase of the protocol, the TA contacts the XCA for issuing the new LDevID certificate. A secure channel (e.g., TLS) protects the communication between these two entities, and the TA uses the DICE LDevID certificate to authenticate itself. For this reason, this phase begins with the TA that retrieves the DICE certificate chain from the SM (2.1/2.3) and provides it to the XCA to validate the client's certificate. Then, after the secure channel handshake (2.4), the TA asks for a nonce (2.5/2.7), which is used to guarantee the freshness of the provided attestation evidence.

As formalised in Section 4.6, the SM signs the TCI of the requesting TA, the LDevID public key for which the certificate is requested, and the nonce received from the XCA. This signature is the attestation evidence inserted in the CSR. Once computed by the SM and returned to the TA, this signature is inserted in the CSR with the nonce. Then the TA requests the SM to sign with the LDevID the CSR (2.8). Finally, the TA transmits the CSR to the XCA (2.9), which will verify its fields as a standard request (2.10). The only additional check is the verification of the attestation evidence. If the nonce in the extension matches the one generated previously, the evidence will be forwarded to the Verifier to check the signature and measurements of the platform. For this check, the XCA contacts Verifier, setting up a secure channel to protect their communication (2.11). The provided data is: the attestation evidence signature, the generated nonce, the LDevID.PK in the CSR, and the DICE certificate chain (2.12). Thus, thanks to the values received and the reference measurements, the Verifier can appraise the DICE chain and the attestation evidence, and inform the XCA about the subject's trustworthiness (2.13/2.14). Finally, if all the checks pass, the new LDevID certificate is issued and sent back to the requesting TA (2.16/2.17). This

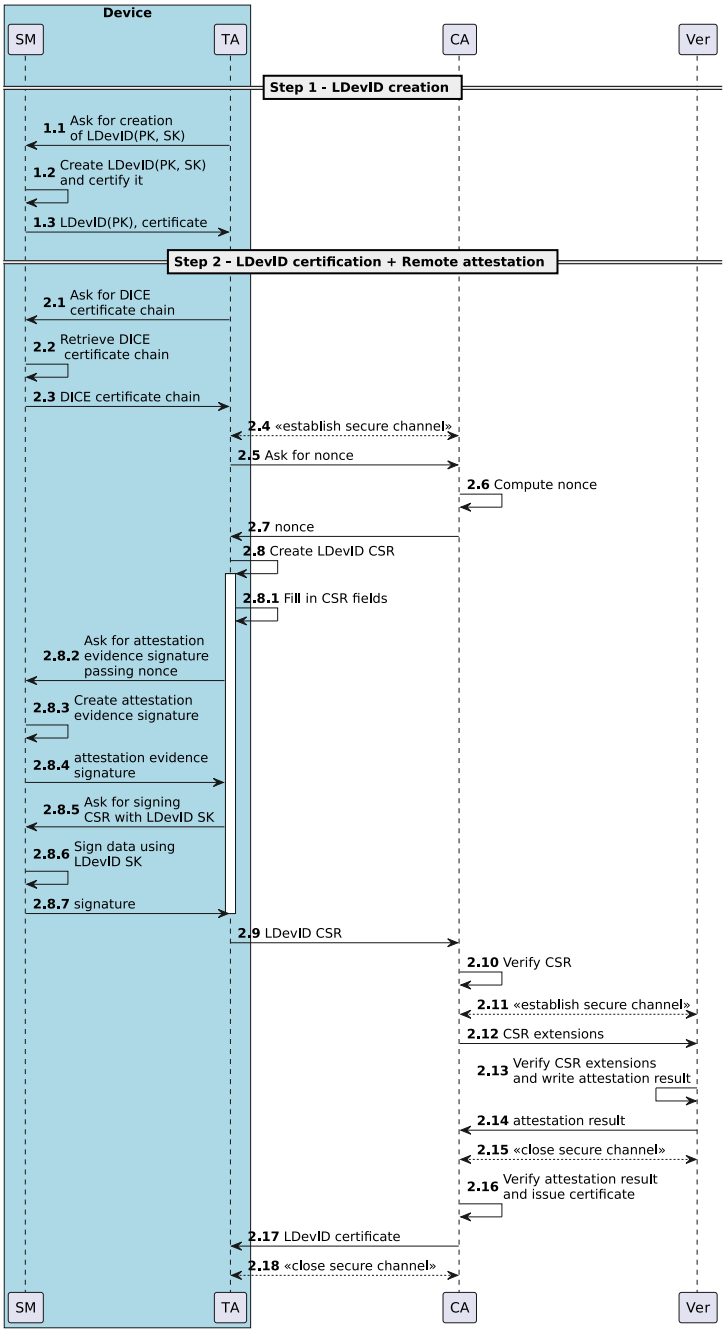


Fig. 4. Certification protocol.

credential can persist; if the component changes, it will not recompute the same key, and old certificates would become useless.

The presented solution ensures the subject's trustworthiness thanks to DICE architecture and TEE protections. As we use the trusted applications of a TEE, compromising them at runtime would require considerable effort from an attacker. Therefore, the trust model is based solely on the application's state at load time, assuming it remains constant throughout execution. In particular, the trusted channel will be set up as a standard secure channel, checking that a trusted XCA issued the certificate and that it is not revoked.

5. Security evaluation

The TEE helps in reducing the attack surface available to adversaries. Since trusted channel endpoints run as TAs, they are isolated from the REE thanks mostly to memory protection and secure I/O. This gives the adversary fewer attack vectors than with standard applications. However, the protection level depends on the TEE that will be chosen to implement this solution. The following analysis is therefore conditional on the assumptions defined in Section 4.1. If the SM or the hardware isolation mechanism is compromised, the adversary could break the binding between the key and the enclave that is using it. This attack is not solved by the certification protocol, but by the correctness of the TEE implementation. For this reason, the analysis below focuses on protocol-level attacks under the stated TEE assumptions.

Runtime attacks have to be distinguished from attacks generated by the untrusted part of the platform. The REE cannot directly modify the enclave memory or access the private key managed by the SM, under the assumptions described in Section 4.1. However, the proposed protocol does not detect a vulnerability inside the TA itself

after the certificate has been issued. The certificate binds the key to the enclave state measured at load time and to the evidence verified during certification. This condition is due to the compliance with the DICE specification, which bases the integrity of the TCB components on load-time measurements [10]. In particular, the specification directly provides guidance for managing DICE-based identities [43], where the security properties can be directly compared with the Trusted Boot process. This property is suitable when the TA is short-lived, or when the credential is used soon after enclave creation. In this case, the load-time measurement and the certified key describe the endpoint during the relevant execution window. If the enclave is expected to run for a long time, the load-time measurement gives weaker assurance about the current integrity status of the endpoint. A possible extension is to bind the certified identity to a key derived from the current executable memory pages of the enclave. The SM could re-measure these pages before authorising the use of the key, or before providing a renewed key corresponding to the new state. In this way, the endpoint could prove that the code memory associated with the certified identity is still the expected one. This would provide a stronger form of run-time code-integrity assurance, while attacks that affect only data or exploit application logic would still require additional protections. This extension requires further support in the TEE and is left as future work.

Considering the certification protocol, the XCA accepts a request only after the validation of the requester's identity and of the related attestation evidence. The certificates of the platform manufacturer or owner can be set as trusted roots, so that the XCA can verify that the request comes from a platform implementing the expected architecture. The DICE certificates bind the identity of the subject and its public key as standard certificates, but also its measurements and configuration. In this way, the Verifier evaluates those values to check that the key has been generated by the expected component. Once this property has been verified, the signatures produced with that key are an implicit attestation of the signer, since only the expected trusted component could have generated and used that key.

Under the assumptions described in Section 4.1, the private keys are not exposed to the untrusted host application. The main remaining threat against the key material is therefore cryptanalysis. This threat is reduced by using different keys for different enclaves and for different purposes, such as authentication, attestation, and certification.

Relay and forwarding attacks are also considered. In this context, the relevant attack is not the simple forwarding of network packets, which is already covered by the secure-channel model. A network adversary may forward packets between two honest endpoints, but, in this case, the real trusted endpoint is still the party that completes the TLS session. The adversary cannot read the protected traffic, modify it without detection, or authenticate as the trusted endpoint without the certified private key.

The critical case for trusted channels is instead the reuse of evidence produced by a trusted platform to certify another key or to authenticate another endpoint. The proposed protocol prevents this attack by binding together the freshness value, the identity key, and the measured application state. The nonce received from the XCA is inserted in the attestation evidence, avoiding the replay of old evidence. The same evidence also includes the LDevID public key for which the certificate is requested. Thus, an adversary cannot replace this key with another one without invalidating the evidence checked by the Verifier. Moreover, the corresponding private key is generated and used through the SM, so it is not available to the untrusted host application, and it cannot be extracted from the SM, under the assumptions described in Section 4.1.

After certification, forwarding only the certificate is not sufficient to impersonate the trusted endpoint. The TLS handshake requires a signature produced with the certified private key. For this reason, the adversary cannot terminate the channel as an attested endpoint unless it compromises the SM, the hardware isolation mechanism, or the trusted application itself, which are the cases discussed in the threat model (Section 4.1).

Finally, the proposed solution satisfies all the security requirements described in Section 4.1. The establishment of the trusted channel implies establishing a standard secure channel (SR1). The node's trustworthiness depends only on the certificate that has been used, so the protocol ensures all the other security properties of the secure channel. Moreover, the linkage between the Remote Attestation session and the secure channel (SR2) is ensured by the key pair used for endpoint authentication. So, the DICE Architecture and the new certification protocol ensure that the endpoint is trustworthy and behaves as expected. Finally, only the XCA and the Verifier are aware of the measurement and configuration of the subject. So, during endpoint authentication, no data is provided about the platform, its layers, and the trusted application. This way, the handshake can be executed without worrying about the leaking of sensitive information (SR3).

6. Implementation

We implemented these new protocols as a proof of concept. The chosen TEE is Keystone [14], a framework to create customisable TEEs on top of RISC-V Hardware [44]. Our previous work [12] has integrated the DICE specifications in Keystone, creating a chain of trust that ends with enclaves. Fig. 5 shows the Keystone layers and their privileges. The first layer is the Trusted Hardware from the bottom, where the DICE HRoT has been implemented. One of the main security controls that RISC-V offers is the PMP. This feature establishes protected memory regions, preventing unauthorised access to the TA's memory. Moreover, the DICE HRoT in [12] provides the secure and measured boot of the SM. Then, this hardware component acts as an ECA, issuing the certificate for the SM's key pair generated according to the DICE Layering Architecture. As introduced previously, the SM is the component that manages the enclave's life cycle and the communication between trusted and untrusted worlds. It runs at the highest RISC-V privilege (Machine mode) and, in the DICE integration, is in charge of generating and operating with the enclaves' LAK and LDevID Key. At the top layer, the TA runs inside an enclave with the lowest RISC-V privilege, signing with its keys through a function of the SM's interface. This secure compartment contains even the Trusted Runtime (RT), a minimal Operating System (OS) that, among its functions, manages the enclave's virtual memory. The instantiation of these trusted components is done in the REE by host applications running on top of a Linux kernel.

In the DICE integration, the certificates issued by ECAs include the subject's measurement in the TCB Info Evidence Extension described in [29, Section 6.1.1]. Furthermore, the implementation of the certification protocol conveys the evidence to the XCA through another DICE extension, named Conceptual Message Wrapper Extension [29, Section 6.1.8].

Secure and trusted channels have been established through a patched version of the Mbed TLS library [45]. We choose it because of its lightweight implementation of TLS, which fits well in an environment with limited resources, such as IoT devices. This patch includes the implementation of the functions to parse and write the DICE extensions described previously, the integration of the SM functions to sign with the public key and other functionalities that were missing in the original implementation (e.g., Ed25519 support, Base64URL encoding and decoding).

Ed25519 is the algorithm used for DICE keys generation, thus the key size is reduced compared to other ciphers such as RSA. Moreover, the performance in key generation and signature computation is suitable for IoT devices with limited capabilities. The new SM's system calls that allow the usage of DICE implementation are:

- `SBI_CREATE_KEYPAIR` generates the enclave's key pair from its CDI and a value received in input;
- `SBI_GET_CHAIN` returns the chain of DICE certificates, i.e., the HRoT's and SM's ECA keys, and the enclave's LAK;
- `SBI_CRYPTO_INTERFACE` wraps the signing function with the enclave's LDevID.

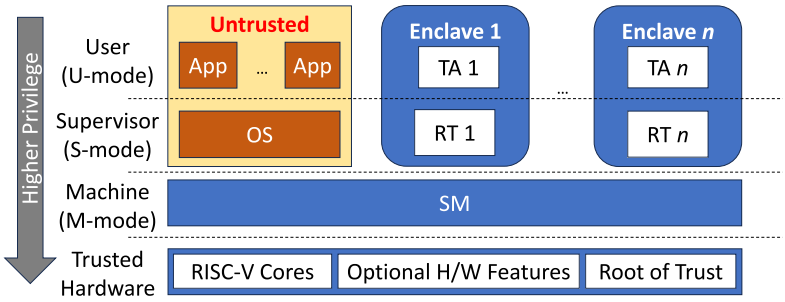


Fig. 5. Stack of components in Keystone system [14].

In the modified version of Keystone, two trusted applications have been implemented to demonstrate the establishment of the proposed trusted channel. The proof of concept also required the definition of two external trusted entities, the XCA and the Verifier.

6.1. Verifier

This node is responsible for evaluating the evidence that the attester provides or the relying party provides in a background check model. One of the functionalities it has to support is allowing external trusted providers to set reference values and policies by which an entity is considered trusted. The Verifier used in the proof of concept is a simple web service that exposes a REST API HTTP POST `/attest` to manage the attestation of an entity: it receives the attestation report from the XCA as request body. It provides the attestation result, i.e., if the entity is evaluated as trusted or not, in the response body. The values inserted in the request include the CSR subject's identity, the generated nonce, the CSR's public key, and the DICE certificate chain. They are necessary for the Verifier to perform the evidence assessment and signature verification. Firstly, the Verifier validates the DICE certificate chain, checking that the certificate corresponding to the DICE HRoT is issued by a trusted manufacturer and, for each certificate, that the measurements inserted in the TCB Info Extension match the golden reference values corresponding to the certificate subject's identity. This check ensures that the LAK, whose certificate is the last of the DICE chain, is a valid attestation key, generated for a trustworthy enclave running on a trusted platform. Then, the Verifier checks the attestation evidence signature using the LAK PK in its certificate. The signed values must match the TCI of the enclave corresponding to the subject's identity (i.e., golden reference value), the nonce sent by the XCA, and the CSR's public key (i.e., LDevID PK). This second check guarantees proof of residency of the enclave's LDevID in the SM of the trusted platform. If all the steps end successfully, the Verifier confirms the requester's trustworthiness in the response.

6.2. External Certificate Authority (XCA)

The main actor in this solution is the XCA. This entity is responsible for verifying the identity and trustworthiness of clients who request new certificates. Delegating the nodes' attestation to a trusted entity reduces the possibility of providing information about the system to untrusted third parties during the channel's creation. The generated credential thus guarantees that the key belongs to a trusted application (at least at load time and at the time of certification). Furthermore, the trusted application manages keys properly. Therefore, any signature made with that key pair implicitly attests the trustworthiness of the node that generated the signature. Suppose the TA running in the enclave is tampered with. In this case, the LDevID key pair provided to the enclave differs. It is cryptographically bound to the enclave's load-time measurement. This binding means that the endpoint cannot prove its trustworthiness. The XCA in the proof of concept is a host application that handles TLS channels, certificates, and CSRs via the patched Mbed TLS library. The Conceptual Message Wrapper extension contains the attestation evidence signature and the associated data as a JSON object in this context. Instead, the TCB Info extension contains the SHA3-512 hash of the component code.

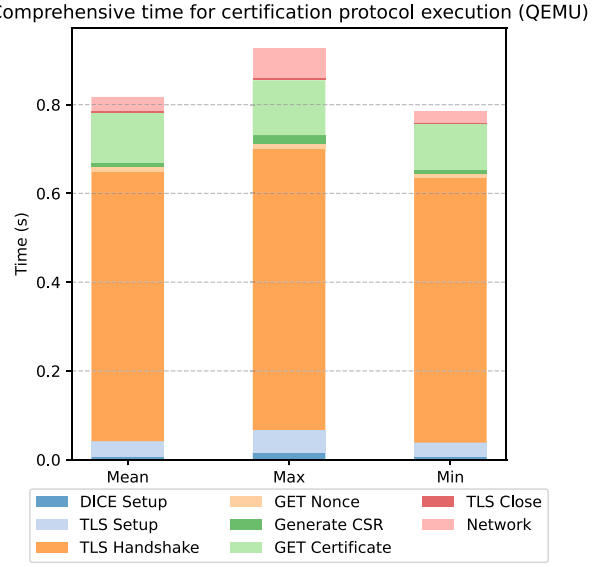


Fig. 6. Execution time for the certification protocol in a TA.

6.3. Trusted channel endpoint applications

In the proof of concept, a client and a server application request the new credential from the XCA and then establish a TLS channel between them. These two applications are TA, so all traffic exchanged between them is protected from the REE, which is only concerned with routing to the enclave. The trusted applications adopt the XCA certificate as a trusted root to validate the credentials provided during the authentication. As a result, neither party knows the platform on which the other is running, making it impossible to identify the manufacturer, owner, or other components. In addition, it is no longer necessary to check two certificate chains as the trust guarantee is managed by the DICE implementation and the XCA, reducing it to a simple certificate validity check. All these features increase the security of information exchange, at the same time leaving the communication protocol similar to a standard secure communication protocol. In fact, the application takes care of obtaining the credential (reading it from memory or obtaining it via the CSR) and then establishing the TLS channel. What is new here are the SM function calls to work with DICE keys and certificates, and the use of the patched version of Mbed TLS. In both cases, few changes are required to the code, making application adaptation quick.

7. Test and validation

Several performance tests were carried out on the proof of concept deployed on a Next Unit of Computing (NUC), equipped with an Intel i5-5300 processor clocked at 2.30 GHz, four cores, eight threads, 16 GB of RAM, and having Ubuntu 22.04 LTS as operating system. The XCA and Verifier applications execute as standard host applications. In contrast, the TLS client and server TAs run in two instances of QEMU [46], an open source machine emulator and virtualiser. Measurements were made of both the execution time of the certification protocol between a TA and the XCA, and the time required to establish a TLS 1.3 channel between the client and server applications. Other tests have been performed with the client TA on QEMU and the server TA on a board, to compare its times with those of the emulator. The device used is a Genesys2 Kintex-7 Field Programmable Gate Array (FPGA) (XC7K325T-2FFG900C) [47] with 1 GB of DDR3 RAM, 256 Mbit Quad serial Flash. It is configured with a CVA6 RISC-V core, which is a 6-stage, single-issue, in-order CPU implementing the 64 bit RISC-V instruction set. We ran each test 32 times, then calculated statistics based on the values obtained, such as min, max, and mean. The raw measurements and the scripts used to process them are publicly available in the project repository.[1] The repository contains the timing traces used for the evaluation and the Python script used to convert hardware ticks into seconds according to the frequency of the corresponding platform.

The evaluation follows the two phases of the proposed approach. The first set of measurements concerns the certification protocol, which is executed before the trusted channel is established. In this phase, the TA creates the LDevID, sets up the TLS connection with the XCA, obtains the nonce, generates the CSR, requests the certificate, and releases the resources used during the exchange. These measurements were collected in the emulated environment and then repeated with the server TA running on the FPGA-based platform. The second set of measurements concerns the channel established after the credential has been issued. In this case, the trusted channel is compared with a secure-channel baseline configured with the same TLS parameters. Finally, we isolate the server-side cost of the certification service by measuring the XCA CSR-validation path under concurrent requests.

Evaluation metrics. The results are reported using different metrics according to the measured phase. Starting from the collected traces, the execution time of each phase was analysed by computing the minimum, maximum, and mean values. For the certification protocol, the mean execution time is denoted as $\overline{L}_{\text{cert}}$. The rate reported in Table 2 is obtained as the inverse of this value:

$$R_{seq} = \frac{1}{\overline{L}_{cert}}$$

This value summarises the sequential cost of one certification execution.

For the comparison between the trusted channel and the corresponding secure-channel baseline, we considered the mean execution time measured after the endpoint credential had been issued. Let $\overline{L}_{TC}$ be the mean execution time of the trusted channel and $\overline{L}_{SC}$ the mean execution time of the secure-channel baseline. The relative overhead is computed as:

$$OH = \frac{\overline{L}_{\text{TC}} - \overline{L}_{\text{SC}}}{\overline{L}_{\text{SC}}} \times 100,$$

where L_{TC} and L_{SC} are the trusted-channel and secure-channel execution times, respectively. Finally, for the XCA benchmark, throughput is computed as:

$$T_{\text{XCA}} = \frac{N_{valid}}{\Delta t},$$

where N_{valid} is the number of CSR validation requests completed during the measurement interval Δt. This last metric refers only to the server-side CSR-validation path.

[1] https://github.com/torsec/test-trust-chan

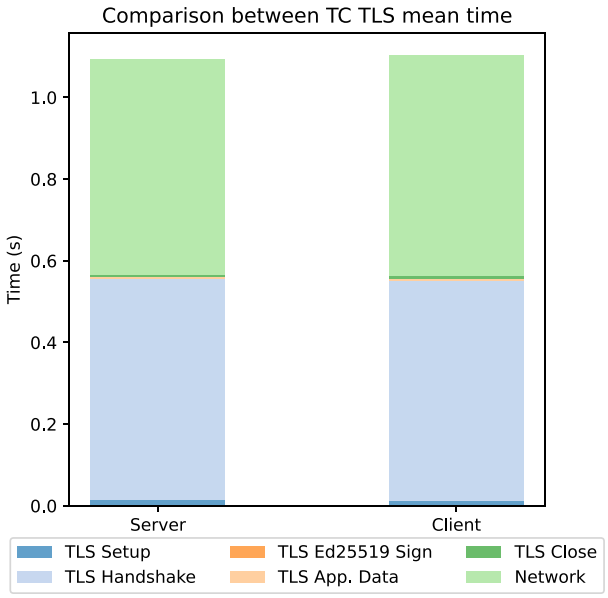


Fig. 7. Mean execution time for the establishment of the trusted channel.

7.1. Performance of certification protocol

Fig. 6 shows the execution time of the certification protocol in QEMU. Firstly, "DICE Setup" involves generating the enclave's LDevID and obtaining DICE certificates, while "TLS Setup" is the time for configuring the TLS channel to the XCA. The longest is taken for the "TLS Handshake", which lasts on average 0.61 s. The connection and packet exchange times, after the TLS Handshake is completed, have been removed and included in the "Network" label to have a value independent of the network. The other times shown in the diagram are for obtaining the nonce ("GET Nonce"), generating the CSR ("Generate CSR") and sending the request to the XCA to receive the new certificate ("GET Certificate"). Wait and network times have not been removed in these cases, as these also depend on the implementations of the XCA and Verifier, and should therefore be regarded as indicative. The last label is "TLS Close" and shows the time for closing the channel and releasing resources.

7.2. Performance of trusted channels between enclave applications

Other tests measured the performance of trusted channels between a client TA and a server TA running on two QEMU instances, using certificates obtained from the XCA for authentication. In this scenario, the XCA issued the certificates signed with a NIST P-521 (`secp521r1`) key pair, and the cipher suite used to set up the TLS channel is `TLS1-3-CHACHA20-POLY1305-SHA256`. Fig. 7 reports the mean time to execute the two TAs. After the TLS handshake, the two applications exchanged short text strings to demonstrate traffic protection. We reported that time as "TLS App. Data", while in "Network", includes the time to send and receive the records over the network. Furthermore, the other labels have the same meaning as in the certification protocol.

Then, a secure TLS channel between two TAs, with the same environment and setting used for the trusted channel, has been established to obtain reference values for the channel establishment phase. In this case, the only differences are the use of random Ed25519 key pairs and the lack of the TCB Info Evidence Extension in the parties' certificates. Fig. 8 compares the mean execution times of a server application in the two cases. The diagram shows the trusted channel performance on the left, while the secure channel performance is on the right. This experiment isolates the trusted-channel execution after

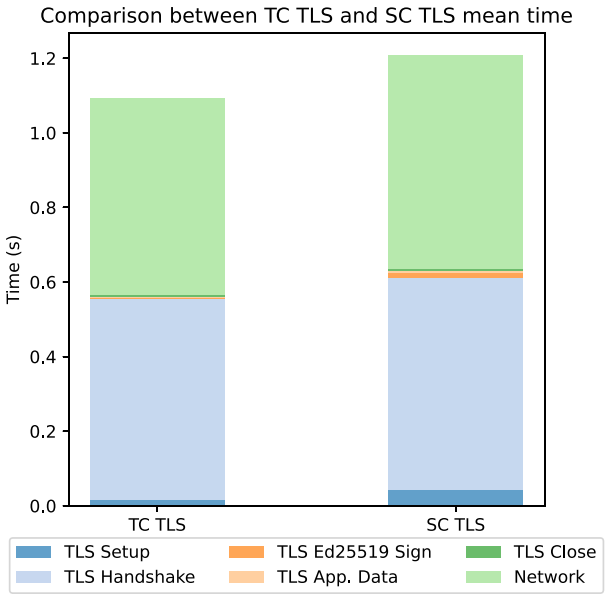


Fig. 8. Comparison between trusted and secure channels' performance.

the endpoint credential has been issued; the certification protocol is evaluated separately. Therefore, the comparison concerns the use of the certified and protected endpoint identity in a standard TLS channel, without exchanging attestation evidence during channel establishment. As expected, the performance of the two channels is virtually identical since there are no modifications to the TLS channel. The slight difference between the values depends only on a few debug prints and the application implementation.

Fig. 8 also shows that protecting the endpoint authentication key through the SM does not introduce a measurable penalty in the tested configuration. In the trusted channel case, the authentication key is the LDevID key generated from the enclave state and used through the SM. In the secure-channel baseline, instead, the authentication key is a random software key generated and used directly by the TA. The trusted-channel execution time is slightly lower in this experiment. This behaviour can be explained by the fact that the signature with the enclave identity key is executed by the SM, which runs in Machine Mode and is not preempted during the operation. The baseline signing operation is executed by the TA, which can be rescheduled. A similar behaviour can be expected in TEE designs where the trusted firmware manages protected keys with higher privilege than enclave applications. However, the exact performance depends on the TEE implementation, on the cost of the transition to the trusted firmware, and on whether the firmware can be extended with the required key-management functions. The relevant result is that, in our prototype, the additional protection of the endpoint identity key does not increase the TLS channel establishment time.

For completeness, we also report an indicative comparison with other trusted channel designs [34,37] in Fig. 9. The comparison has to be considered only as a high-level indication, since the solutions were evaluated in different environments and with different implementations. However, it helps to show the effect of moving the attestation check outside the channel establishment phase. In our prototype, the overhead relative to the secure-channel baseline is 3.5%. The other approaches report a larger overhead, since attestation evidence is transported and verified during the establishment of the protected channel.

7.3. Performance of the certification protocol on a physical device

Performance tests measured the mean execution time of the server TA on the board, which communicated with the other applications deployed as in the previous tests. Fig. 10 reports the values for the certification protocol, which are much longer than the runs in QEMU. On average, the protocol required 0.817 s in the emulator and 27.2 s in the board, which is more than 33 times longer. In this test, the worst slowdown was in the TLS handshake (4269%), as this is the time that depends the most on the execution of the code in the TA, while the others are communication times with the XCA executed on the NUC. A reduction in execution times could be achieved by using a higher-end FPGA, with greater computational power than Genesys2.

Table 2
Latency-derived rates for the complete certification protocol.

Scenario	Mean time (s)	Derived rate (executions/s)
TA in QEMU	0.817	1.22
TA on board	27.2	0.037

Table 2 reports the rates obtained as the inverse of the mean execution time of the complete certification protocol. These values provide an indication of the sequential cost of one certification execution, but they are not obtained from a concurrent load test. Therefore, they characterise the latency-derived rate of a single requester and do not represent the end-to-end throughput that a deployment would sustain with multiple concurrent trusted endpoints. The large difference between QEMU and the board is mainly due to the limited computational capability of the FPGA platform, which also affects the TLS handshake phase.

7.4. Server-side throughput of the XCA CSR-validation path

To complement the latency measurements, we isolated the server-side part of the certification service by evaluating the XCA during the validation of enclave CSRs. In this benchmark, concurrent workers submit already generated CSRs containing attestation material, and the XCA performs the checks required before certificate issuance. The test was executed with 2, 4, 8, 16, and 32 concurrent workers, and throughput was measured as the number of completed validation requests per second.

As reported in Table 3 and Fig. 11, the XCA benefits from moderate concurrency. Throughput increases from 172.23 requests/s with 2 threads to 332.11 requests/s with 4 threads, reaching 415.14 requests/s with 8 threads. With 16 and 32 threads, throughput decreases to 267.09 and 229.85 requests/s, respectively. This behaviour indicates that, on the machine used in our experiments, the validation path scales up to a moderate level of parallelism before becoming affected by contention and scheduling overhead.

These results have to be interpreted together with the latency measurements reported above. The XCA throughput test measures only the validation path of the certification service, while the complete certification protocol also includes enclave-side execution, TLS setup, TLS handshake, CSR generation, message exchanges, certificate retrieval, and resource release. This distinction is important because the XCA and the Verifier run on the same host machine in both the QEMU and board experiments, whereas the trusted endpoint runs on two platforms with very different computational capabilities. The complete certification protocol takes about 0.817 s in QEMU and 27.2 s on the board, although the server-side certification components are unchanged. This difference indicates that, in the tested configuration, the XCA is not the limiting factor of the protocol. The main cost is instead due to the execution of the endpoint-side operations inside the trusted environment, and this cost becomes dominant on the constrained FPGA platform.

The present evaluation does not include controlled Round-Trip Time (RTT) or packet-loss emulation. The experiments therefore characterise the local processing cost of the prototype and the scalability of the XCA validation path, but they do not provide a complete Wide Area Network (WAN) sensitivity analysis. This choice follows the scope

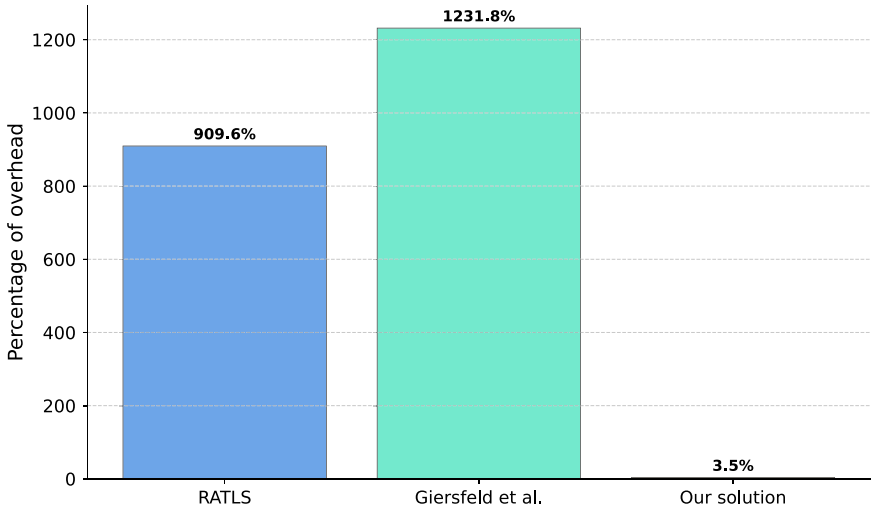


Fig. 9. Indicative comparison of the overhead introduced by attestation-based trusted channel approaches. Values for the external approaches are derived from the corresponding published results. The overhead is computed as the relative increase over the corresponding secure-channel baseline reported for each solution.



Fig. 10. Execution time for the certification protocol in a TA on a board.

Table 3
XCA throughput during enclave CSR validation.

Concurrency	Throughput (requests/s)
2 threads	172.23
4 threads	332.11
8 threads	415.14
16 threads	267.09
32 threads	229.85

of the work. After certificate issuance, the trusted channel is established as a standard TLS 1.3 channel, without carrying attestation evidence during the handshake. Therefore, RTT and packet loss affect the communication phase as in the corresponding TLS deployment. The additional cost introduced by the proposal is concentrated before the channel establishment, during certificate request and issuance. For this reason, the evaluation focuses on the latency of this phase, on the throughput of the XCA validation path, and on the comparison with a secure-channel baseline where the authentication keys are generated and used directly by the TA. The results show that the use of SM-protected identity keys does not add measurable overhead to the TLS establishment in the tested prototype. A systematic evaluation under controlled network conditions is left as future work.

Table 4
Summary of the performance evaluation.

Measured aspect	Setting	Main value	Interpretation
Certification protocol	QEMU	0.817 s / 1.22 exec/s	Sequential credential issuance; rate derived from latency.
Certification protocol	Board	27.2 s / 0.037 exec/s	Sequential credential issuance on the physical platform.
Post-certification trusted channel	QEMU	3.5% overhead	TLS establishment after certificate issuance.
XCA CSR-validation path	XCA host	415.14 req/s at 8 workers	Concurrent server-side validation only.

Table 4 summarises the scope of the measurements reported in this section.

8. Conclusion and future work

We defined a solution to integrate the Remote Attestation of the platform by establishing a secure channel. The work creates a new certification protocol. A TA can obtain a credential from an external trusted third party, and an X.509 certificate for an Ed25519 key pair generated according to the DICE specifications. Many factors ensure the trustworthiness of the party. Firstly, the DICE architecture ensures that the key used to establish the secure channel is generated only by an application running in an enclave of the TEE in a specific device. The private part of the key is managed by the SM and is not exported to the TA. The TA can only request cryptographic operations through the interface provided by the SM. This design choice preserves the binding between the enclave measurement and the actual endpoint identity, preventing the identity key from being migrated together with the enclave data. Then, the attestation performed before certificate issuance binds the identity of the TA to the LDevID public key included in the certificate. Therefore, a successful use of the certified key during the TLS handshake provides evidence that the endpoint corresponds to the trusted application state verified at certification time. Moreover, executing the secure channel endpoint in a trusted application reduces

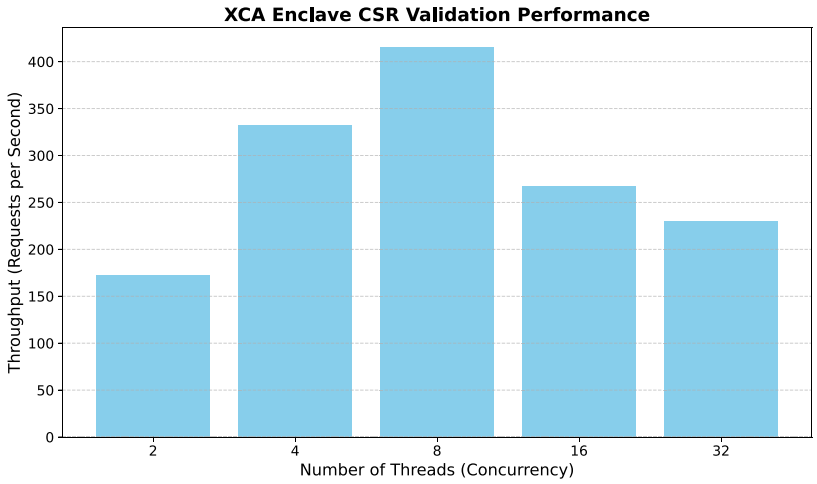


Fig. 11. XCA throughput during enclave CSR validation under concurrent requests.

the attack surface. It increases the attacking complexity due to the security controls offered by TEE.

Using these new credentials to establish the secure channel provides additional security guarantees about the endpoint while keeping the subsequent communication based on a standard TLS channel. In our prototype, once the endpoint credential has been issued, the channel establishment time is comparable with the corresponding secure-channel baseline. The overhead introduced by the proposal is concentrated before channel establishment, during certificate request and issuance, and by the execution of the endpoint inside the TEE. The certification cost can be amortised over the lifetime of the issued credential, while the subsequent communication remains based on a standard TLS channel. However, these two features belong to several contexts and use cases, so the proposed solution requires only modifying existing components. Moreover, the architecture of the XCA and verifier, or their APIs, can be modified based on the scenario and the product, but without changing their purpose and key elements.

Future work includes extending the threat model to consider the TA compromise at runtime. A possible solution to this problem could be to adopt existing works to include attestation proofs in the TLS handshake, such as [28]. Thus, the endpoint is aware of the other party's trustworthiness when establishing the protected communication. In addition, the protocol should be formally verified to demonstrate compliance with the security requirements and to attempt to find possible attacks against it. Among protocol verifiers, Proverif [48] and Tamarin [49] have been considered to carry out security verification. Another complementary direction is the adoption of TEE implementations with stronger assurance. This includes the formal verification of isolation mechanisms and the reduction of the SM attack surface, as investigated in recent works on RISC-V TEEs [42,50]. In conclusion, the performance obtained justifies continuing this work and integrating it into environments and protocols other than TLS.

CRediT authorship contribution statement

Giacomo Bruno: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Silvia Sisinni:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Data curation, Conceptualization. **Enrico Bravi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Data curation, Conceptualization. **Lorenzo Ferro:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Flavio Ciravegna:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Data curation, Conceptualization. **Antonio Lioy:** Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Antonio Lioy reports financial support was provided by European Commission. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805, http://dx.doi.org/10.1016/j.comnet.2010.05.010.

[2] E. Bravi, A. Lioy, Securing IoT devices: an overview, in: IEEE Symposium on Computers and Communications, ISCC, IEEE, Bologna (Italy), 2025.

[3] M. Nawir, A. Amir, N. Yaakob, O.B. Lynn, Internet of Things (IoT): Taxonomy of security attacks, in: 2016 3rd International Conference on Electronic Design, ICED, Phuket (Thailand), 2016, pp. 321–326, http://dx.doi.org/10.1109/ICED.2016.7804660.

[4] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, 2018, http://dx.doi.org/10.17487/RFC8446, RFC-8446.

[5] K. Seo, S. Kent, Security Architecture for the Internet Protocol, 2005, http://dx.doi.org/10.17487/RFC4301, RFC-4301.

[6] A. Seshadri, A. Perrig, L. van Doorn, P. Khosla, SWATT: softWare-based attestation for embedded devices, in: IEEE Symposium on Security and Privacy, Berkeley (CA, USA), 2004, pp. 272–282, http://dx.doi.org/10.1109/SECPRI.2004.1301329.

[7] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, P. Khosla, Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems, in: Twentieth ACM Symposium on Operating Systems Principles, Brighton (United Kingdom), 2005, pp. 1–16, http://dx.doi.org/10.1145/1095810.1095812.

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

[9] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted Execution Environment: What It is, and What It is Not, in: IEEE Trustcom/BigDataSE/ISPA, Vol. 1, Helsinki (Finland), 2015, pp. 57–64, http://dx.doi.org/10.1109/Trustcom.2015.357.

[10] Trusted Computing Group, DICE Layering Architecture, 2020, https://trustedcomputinggroup.org/wp-content/uploads/DICE-Layering-Architecture-r19_pub.pdf.

[11] E. Bravi, S. Sisinni, A. Lioy, Exploiting the DICE specification to ensure strong identity and integrity of IoT devices, in: 8th International Conference on Smart and Sustainable Technologies, SpliTech, Split/Bol (Croatia), 2023, http://dx.doi.org/10.23919/SpliTech58164.2023.10193517.

[12] 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, http://dx.doi.org/10.1109/ACCESS.2025.3596829.

[13] S. Sisinni, D.G. Berbecaru, V. Donnini, A. Lioy, MATCH-IN: Mutual Attestation for Trusted Collaboration in Heterogeneous IoT Networks, in: IEEE Symposium on Computers and Communications, ISCC, Paris (France), 2024, pp. 1–6, http://dx.doi.org/10.1109/ISCC61673.2024.10733616.
[14] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, D. Song, Keystone: An Open Framework for Architecting Trusted Execution Environments, in: Fifteenth European Conference on Computer Systems, Heraklion (Greece), 2020, pp. 1–16, http://dx.doi.org/10.1145/3342195.3387532.
[15] TCG, Trusted Computing Group, 2025, https://trustedcomputinggroup.org/.
[16] TCG, Trusted Platform Module Library Part 1: Architecture, 2024, https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-1.83-Part-1-Architecture.pdf.
[17] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, B. Sniffen, Principles of remote attestation, Int. J. Inf. Secur. 10 (2) (2011) 63–81.
[18] V. Costan, S. Devadas, Intel SGX Explained, 2016, Cryptology ePrint Archive, Paper 2016/086, https://eprint.iacr.org/2016/086.
[19] K. Eldefrawy, N. Rattanavipanon, G. Tsudik, HYDRA: Hybrid Design for Remote Attestation (Using a Formally Verified Microkernel), in: 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Boston (MA, USA), 2017, pp. 99–110, http://dx.doi.org/10.1145/3098243.3098261.
[20] K. Eldefrawy, G. Tsudik, A. Francillon, D. Perito, Smart: secure and minimal architecture for (establishing dynamic) root of trust, in: 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Boston (MA, USA), 2017, pp. 99–110.
[21] www.globalplatform.org | © 2011-2022 GlobalPlatform, Inc. GlobalPlatform Technology, TEE System Architecture v1.3, 2022, https://globalplatform.org/wp-content/uploads/2022/05/GPD_SPE_009-GPD_TEE_SystemArchitecture_v1.3_PublicRelease_signed.pdf.
[22] 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, 2016, pp. 445–451, http://dx.doi.org/10.1109/CIC.2016.065.
[23] C. Shepherd, R.N. Akram, K. 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), 2017, http://dx.doi.org/10.1145/3098954.3098971.
[24] Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, N. Asokan, Beyond secure channels, in: ACM Workshop on Scalable Trusted Computing, Alexandria (Virginia, USA), 2007, pp. 30–40, http://dx.doi.org/10.1145/1314354.1314363.
[25] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, D. Cooper, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008, http://dx.doi.org/10.17487/RFC5280, RFC-5280.
[26] F. Armknecht, Y. Gasmi, A.-R. Sadeghi, P. Stewin, M. Unger, G. Ramunno, D. Vernizzi, An efficient implementation of trusted channels based on openssl, in: 3rd ACM Workshop on Scalable Trusted Computing, Alexandria (Virginia, USA), 2008, pp. 41–50, http://dx.doi.org/10.1145/1456455.1456462.
[27] IETF, Remote ATtestation ProcedureS Working Group, 2025, https://datatracker.ietf.org/group/rats/about/.
[28] H. Tschofenig, Y. Sheffer, P. Howard, I. Mihalcea, Y. Deshpande, A. Niemi, T. Fossati, Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), 2024, URL https://datatracker.ietf.org/doc/draft-fossati-tls-attestation/08/.
[29] Trusted Computing Group, DICE Attestation Architecture, 2024, https://trustedcomputinggroup.org/wp-content/uploads/DICE-Attestation-Architecture-Version-1.1-Revision-18_pub.pdf.
[30] G. King, H. Wang, HTTPA: HTTPS Attestable Protocol, 2022, http://dx.doi.org/10.48550/arXiv.2110.07954.
[31] G. King, H. Wang, HTTPA/2: a Trusted End-to-End Protocol for Web Services, 2022, http://dx.doi.org/10.48550/arXiv.2205.01052.
[32] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, M. Vij, Integrating Remote Attestation with Transport Layer Security, 2019, http://dx.doi.org/10.48550/arXiv.1801.05863.
[33] R. Barnes, J. Hoffman-Andrews, D. McCarney, J. Kasten, Automatic Certificate Management Environment (ACME), 2019, http://dx.doi.org/10.17487/RFC8555, RFC-8555.
[34] R. Walther, C. Weinhold, M. Roitzsch, RATLS: Integrating Transport Layer Security with Remote Attestation, in: Applied Cryptography and Network Security Workshops, ACNS, Rome (Italy), 2022, http://dx.doi.org/10.1007/978-3-031-16815-4_20.
[35] OpenSSL Software Foundation Inc, OpenSSL, https://www.openssl.org/.
[36] C. Weinhold, M. Roitzsch, RATLS: Remote Attestation Integrated with Transport Layer Security, https://github.com/Barkhausen-Institut/ratls.
[37] P. Giersfeld, Establishing Trusted Channels for Confidential Workloads (Master's thesis), Aalto University, Espoo, Finland, 2024, https://urn.fi/URN:NBN:fi:aalto-202409016132.
[38] J.A. Donenfeld, WireGuard, https://www.wireguard.com/.
[39] Confidential Containers Contributors, Confidential Containers: Deploy Cloud Native Applications inside Confidential Enclaves, https://confidentialcontainers.org/.
[40] H. Birkholz, D. Thaler, M. Richardson, N. Smith, W. Pan, Remote ATtestation procedureS (RATS) Architecture, 2023, http://dx.doi.org/10.17487/RFC9334, RFC-9334.
[41] B. Weeks, Automated Certificate Management Environment (ACME) Device Attestation Extension, 2024, https://datatracker.ietf.org/doc/draft-acme-device-attest/03/.
[42] K. Cheang, C. Rasmussen, D. Lee, D.W. Kohlbrenner, K. Asanović, S.A. Seshia, Verifying RISC-V Physical Memory Protection, 2022, http://dx.doi.org/10.48550/arXiv.2211.02179.
[43] Trusted Computing Group, Implicit Identity Based Device Attestation, 2018, https://trustedcomputinggroup.org/wp-content/uploads/TCG-DICE-Arch-Implicit-Identity-Based-Device-Attestation-v1-rev93.pdf.
[44] RISC-V International, RISC-V: The Open Standard RISC Instruction Set Architecture, 2025, https://riscv.org/.
[45] MBed TLS, MBed TLS, 2024, https://mbed-tls.readthedocs.io/.
[46] QEMU, QEMU, 2025, https://www.qemu.org/.
[47] Genesys 2 FPGA Board Reference Manual, 2017, https://digilent.com/reference/_media/reference/programmable-logic/genesys-2/genesys2_rm.pdf.
[48] B. Blanchet, V. Cheval, Proverif, 2025, https://bblanche.gitlabpages.inria.fr/proverif/.
[49] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, B. Schmidt, Tamarin Prover, 2025, https://tamarin-prover.com/.
[50] M. Kuhne, S. Volos, S. Shinde, Dorami: Privilege Separating Security Monitor on RISC-V TEEs, in: 34th USENIX Security Symposium, Seattle (WA, USA), 2025, https://www.usenix.org/system/files/usenixsecurity25-kuhne.pdf.