Universal Remote Attestation for Cloud and Edge Platforms

Simon Ott
Fraunhofer AISEC
Garching near Munich, Germany
simon.ott@aisec.fraunhofer.de

Monika Kamhuber
Fraunhofer AISEC
Garching near Munich, Germany
monika.kamhuber@aisec.fraunhofer.de

Joana Pecholt
Fraunhofer AISEC
Garching near Munich, Germany
joana.pecholt@aisec.fraunhofer.de

Sascha Wessel
Fraunhofer AISEC
Garching near Munich, Germany
sascha.wessel@aisec.fraunhofer.de

ABSTRACT

With more computing workloads being shifted to the cloud, verifying the integrity of remote software stacks through remote attestation becomes an increasingly important topic. During remote attestation, a *prover* provides *attestation evidence* to a *verifier*, backed by a hardware trust anchor. While generating this information, which is essentially a list of hashes, is easy, examining the trustworthiness of the overall platform based on the provided list of hashes without context is difficult. Furthermore, as different trust anchors use different formats, interaction between devices using different attestation technologies is a complex problem.

To address this problem, we propose a universal, hardware-agnostic device-identity and attestation framework. Our framework focuses on easing attestation by having provers present meaningful metadata to verify the integrity of the attestation evidence. We implemented and evaluated the framework for Trusted Platform Modules (TPM), AMD SEV-SNP attestation, and ARM PSA Entity Attestation Tokens (EATs).

CCS CONCEPTS

• **Security and privacy** → *Security protocols*; *Trusted computing*; *Distributed systems security*; *Embedded systems security*; *Virtualization and security*.

KEYWORDS

Remote Attestation, Integrity Validation, Channel Binding, TPM, AMD SEV-SNP, ARM PSA

ACM Reference Format:
Simon Ott, Monika Kamhuber, Joana Pecholt, and Sascha Wessel. 2023. Universal Remote Attestation for Cloud and Edge Platforms. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3600160.3600171

ARES 2023, August 29–September 01, 2023, Benevento, Italy

ACM ISBN 979-8-4007-0772-8/23/08.
https://doi.org/10.1145/3600160.3600171

1 INTRODUCTION

Cloud computing has become prevalent and continues to increase as more enterprises shift data and software to the cloud [20]. As this software and data is running on machines controlled by remote parties, establishing trust in the remote cloud services is essential. At the same time, an increasing number of Internet of Things (IoT) devices is connected, requiring trustworthiness examination as well, as attacks on IoT devices are widespread [19].

Trust in remote devices can be established via attestation, which is the identification, authentication, and software integrity verification of a platform [11]. Remote attestation requires an attesting or proving platform to provide *measurements* of the software stack to a remote verifying party. Each entity may take on both roles to achieve mutual attestation. To avoid tampering or forging the measurements, these are generated and signed by a hardware trust anchor. This data structure, the *attestation evidence* is presented to a verifying party for examination. To securely exchange data, the remote attestation protocol can further be bound to a secure channel through channel binding.

Hardware technologies like Trusted Platform Modules (TPMs), AMD Secure Encrypted Virtualization (SEV)-Secure Nested Paging (SNP) (SEV-SNP), Intel Trust Domain Extensions (TDX), or Arm TrustZone provide capabilities for acting as such a trust-anchor and providing attestation evidence for verification. Those technologies are available on many computing platforms.

However, remote attestation functionalities often remain unused. One of the reasons is that it is difficult for the verifying party to examine the trustworthiness of the overall software stack based on the attestation evidence from the attesting platform [10], because this information mainly is an aggregated hash or list of hashes of software, which is difficult to interpret. Furthermore, there is no generic format or framework supporting different technologies and, thus, no hardware-agnostic remote attestation mechanism.

Generic architectures, such as Remote ATtestation ProcedureS (RATS) [11] try to solve the problem by splitting the verifying party into two roles, a *verifier* and a *relying party*. As the verification process is deemed too complex for the relying party, the verification is offloaded to the *verifier*. The verifier is dependent on further roles: an *endorser* and a *reference value provider*, which provide the required metadata to examine the trustworthiness of the *evidence* supplied by the *attester*. This architecture is shown on the left side of Figure 1. The drawback of such architectures is, that the relying party is dependent on a verifier, which requires an additional trust relationship between these two parties. Furthermore, the verifier

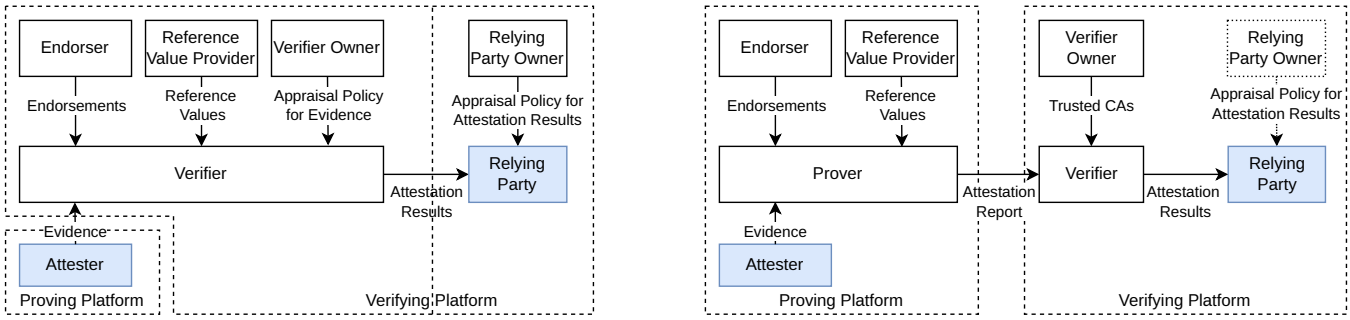

Figure 1: Mapping between generic architecture (left) [11] and our framework (right)

requires knowledge of all legitimate software that might be running on an attesting platform. Such a list might be very large and difficult to obtain. This paper therefore presents a different approach, aiming for an easier verification of software stacks for the verifying party. While not limited to, our framework especially targets the verification of previously unknown software stacks of different vendors.

We achieve this through shifting the burden of providing meaningful metadata for assessing the trustworthiness of the measurements to the proving platform, as shown on the right side of Figure 1. This avoids the relying party's dependency on a *Reference Value Provider* and an *Endorser* supplying them with metadata for evaluating the trustworthiness of the measurements. We consider it feasible on the proving platform to compile a list of reference values, i.e., to perform the tasks of the verifier as defined in [11], as the software running on the proving platform is known to its operator.

We define the format for an attestation report as the minimal set of linked metadata required to describe the desired state of a computing platform. To establish trust in the prover supplied metadata, i.e., reference values and endorsements, each of those elements must be signed by one or multiple common Roots of Trust (RoTs). Thus, trust in unknown remote computing platforms can be established solely based on one or multiple common RoTs without any additionally supplied information on the verifier side. To allow flexibility in different scenarios, our format allows adding further metadata on the prover side and configuring a generic policy engine on the verifier side.

The result is a universal device-identity and attestation framework, supporting attestation between different devices and technologies. Our approach forces providers of computing platforms, such as cloud or Virtual Private Network (VPN) providers to know and understand their software stack, while it makes it a lot simpler for consumers (e.g. companies buying cloud computing power) to establish trust in the remote software stacks before processing or storing sensitive data.

To prove the feasibility of our solution, we implemented a Proof of Concept (PoC) attestation daemon which can act as verifier as well as prover and supports TPM-based, AMD SEV-SNP-based as well as ARM PSA-based attestation. We furthermore provide a library for establishing secure channels to remote computing platforms bound to the remote attestation. We evaluate our solution regarding performance and overhead, as well as by conducting a conceptual security analysis.

2 BACKGROUND

Our concept requires a hardware trust anchor for generating and storing cryptographic keys and attesting the current state of the platform. We provide required background for the trust anchors used in our PoC in the following.

2.1 Trusted Platform Module (TPM)

A TPM is a secure crypto-processor designed to enable trust in computing platforms. TPMs can generate and store cryptographic keys, store measurements of software artifacts, perform cryptographic operations and attest the current state of the platform.

Keys and Certificates. TPMs can create and store keys in different key hierarchies. Furthermore, TPMs are shipped with an Endorsement Key (EK) and corresponding certificate, which is part of a certificate chain with a manufacturer's Certificate Authority (CA) as the root. Attestation Keys (AKs) can be derived from the EK for signing operations. In order to link the AK to the EK, a challenge-response-protocol called the TPM *Credential Activation* is performed.

Measurements and Chain of Trust. TPMs provide Platform Configuration Registers (PCRs) that are utilized as storage for integrity measurements. The size of one PCR is exactly the size of the hash-algorithm used for the measurements. The lower 16 PCRs are static, i.e., initialized to all zeros on platform reset. Afterwards, they cannot be reset but only be *extended* via the following operation:

$$PCR[n] = Hash(PCR[n]||ArgumentOfExtend) \quad (1)$$

ArgumentOfExtend is the software component to be measured. PCRs 17 and up are dynamic, initialized to all ones and can only be reset to zeros by certain privileged CPU instructions. They can also be extended as defined in Equation (1).

To establish trust, a device can use a TPM to perform a *Measured Boot*: Starting from an implicitly trusted Core Root of Trust for Measurements (CRTM), each component measures the next component before transferring control. This builds a chain of trust starting at the CRTM.

Furthermore, two different approaches exist for the Measured Boot: Static Root of Trust for Measurements (SRTM) [23] and Dynamic Root of Trust for Measurements (DRTM) [21]. The SRTM approach comprises all components of the software stack and is performed during boot. The DRTM approach requires dedicated hardware and is performed during runtime: Upon request, the CPU switches to a secure state and triggers the execution of a *Secure Loader* measuring the runtime components into the dynamic PCRs, omitting the early boot components from the Trusted Computing Base (TCB). Both approaches can be used for remote attestation: The TPM can produce a structure containing metadata about the system state including the PCRs, called Quote. The Quote is signed by an AK and comprises the content of selected PCRs defining the measured software stack and user supplied data.

2.2 AMD SEV-SNP

Confidential Computing (CC) aims to protect data at rest, in use and in transit from unauthorized access and provide remote attestation capabilities. Hardware-based technologies are used to implement such CC environments. One example is AMD's SEV-SNP technology, which is designed to protect Virtual Machines (VMs) from the hypervisor [26]. SNP encrypts and integrity-protects memory and CPU state of the VM through hardware mechanisms and the SNP firmware [2]. AMD provides remote attestation capabilities to verify these SNP-protected VMs.

Keys and Certificates. AMD introduced a Public-Key Infrastructure (PKI) to facilitate remote attestation of SNP-protected VMs. The *AMD Root Key* pair defines hardware RoT and signs the *AMD Signing Key* certificate. This key pair in turn signs the Versioned Chip Endorsement Key (VCEK) certificate, which is unique to every platform and firmware version. Its private key cannot be extracted from the hardware and is used by the SEV firmware to sign the attestation report generated for a specific VM.

Measurements and Chain of Trust. To remotely attest an SNP-protected VM, the attestation report must contain information on the software stack that is running inside the VM. The SEV firmware measures all components that are loaded and encrypted into the VM's memory before it is started. This initial measurement is then stored and used to create attestation reports for the VM at runtime [3]. The measurement typically encompasses the Unified Extensible Firmware Interface (UEFI) and possibly further components. To ensure that the later components of the software stack are trusted or attested, secure boot or measured boot features must be used.

2.3 Arm PSA Attestation Token

Resource-constrained Arm Cortex-M devices beginning with the ARMv8-M architecture can implement the Arm TrustZone technology [1]. The TrustZone technology is a hardware-based isolation built into the CPU. On a TrustZone-enabled processor, the CPU runs either in a secure or a non-secure state. Resources such as memory, interrupts or peripherals can be configured to be either secure or non-secure and the CPU operating in non-secure mode cannot access secure resources. This enables the design of Trusted Execution Environments (TEEs), which can be used for hybrid attestation mechanisms, with a hardware-enforced isolation of the software measurements during the measured boot. For attestation, ARM provides the Platform Security Architecture (PSA) attestation token. The PSA attestation token specification is part of a generic Internet Engineering Task Force (IETF) working group[1] approach towards remote attestation, the RATS architecture.

Keys and Certificates. The Initial Attestation Service (IAS) uses an asymmetric key-pair to sign a metadata structure (comparable to a TPM quote). The metadata structure contains the software measurements for the platform, verifier supplied data, i.e., a nonce, called Auth Challenge, and further information about the platform.

Measurements and Chain of Trust. Trusted Firmware-M with IAS performs a measured boot and records the measurements within secure storage in the TEE. Trusted Firmware-M supports the use of a secure bootloader, such as MCUBoot. The CPU starts in secure mode and measures the runtime firmware. Depending on the device, the integrity of the bootloader can be protected via flash locks or a secure boot can be implemented in the boot ROM.

3 THREAT MODEL AND REQUIREMENTS

The TCB comprises the hardware trust anchor, the CPU and firmware running on the machine, any software required for measuring and enforcing the validity of software running on the next stage in the boot chain, as well as any software being able to access or modify sensitive data on the machine. The specific TCB components depend on the underlying hardware trust anchor, e.g., on an SNP machine, the host kernel or hypervisor and host user space software are not part of the TCB [27]. In contrast, on a TPM-based machine, those components are part of the TCB.

We assume an attacker with Dolev-Yao attacker model [17] capabilities, i.e., they are a legitimate communication peer inside the network and can replay, read or modify any transmitted data as well as relay data to other end-points. Furthermore, the attacker has remote access to the machine with root privileges. This is called an *Inside Attacker* [39] and could, e.g., be an administrator in a cloud context.

The following limitations apply to the attacker. We assume that software, which the verifier expects to run on the platform, behaves as expected, i.e., does not contain any vulnerabilities. This is a general limitation of remote attestation: It can only ensure that a platform runs exactly the software that is expected, but not that this software is secure by itself. We assume the hardware works as specified, the attacker is not able to break our TCB, and the attacker is not able to break state of the art cryptographic ciphers or protocols. Furthermore, physical attacks and side channel attacks on the utilized hardware are out of scope, as are attacks on the CAs issuing identity certificates, and Denial of Service (DoS) attacks.

The main goal of our attacker is either to compromise the software stack of a remote machine without being detected, or to impersonate a trustworthy machine, e.g., to get access to sensitive data. In view of our threat model and based on [32], we define the following security requirements for our design:

REQ-I (Secure Channel) Confidentiality, integrity and authenticity of all data transferred between attested devices

[1] https://datatracker.ietf.org/wg/rats/about/

REQ-II (Trustworthiness) Trustworthiness (integrity) of the software stack of the remote computing platform

REQ-III (Channel Binding) Binding of attestation evidence to the secure channel end-point

REQ-II requires that the verifier can assess that expected, i.e., trusted, software is running on the remote platform (**REQ-IIa**) and that this software is up-to-date (**REQ-IIb**).

4 CONCEPT

In this section, we present the design of our identity and attestation framework for verifying the trustworthiness of remote and unknown computing platforms. At its core is the *attestation report*, a self-contained data structure generated by the prover and validated by the verifier. This *attestation report* must contain all relevant information to make assertions about the identity and trustworthiness of a previously unknown remote computing platform solely based on a set of common trust anchors. We therefore define the attestation report as the signed minimal set of linked metadata required to describe the desired state of a computing platform and make claims about its trustworthiness. We enforce this, because we do not consider it feasible that relying parties have access to verifiers supplying them with up-to-date endorsements, reference values and appraisal policies for all computing platforms they want to potentially communicate with. In contrast, the operator of the proving platform knows exactly the trust anchor and software they deployed on the platform, making it an acceptable burden to provide a minimal set of signed metadata required to make claims about the platform.

Every attestation report metadata object must be signed by a trustworthy entity and the attestation report must contain all certificate chains, so that the verifier can verify the signatures and establish trust in the signing certificates via the underlying PKI.

Thus, a verifier only requires the attestation report and a list of trusted CAs from the verifier owner and can perform the attestation without additionally supplied information. This avoids carrying a large allowlist of all possible valid software measurements on the verifier side. Instead, trust can be established solely based on one or multiple common RoTs.

The basic remote attestation process is as follows: The prover interacts with the utilized hardware trust anchor to fetch the protected measurements. Doing this, the prover provides a nonce chosen by the verifier to be included in the measurements for replay protection. The prover combines these with the metadata describing the device and its software to an attestation report. The verifier receives and processes the attestation report. For this, the verifier owner only has to specify a set of CAs it trusts. This way, the verifier verifies the measurements via their signatures and certificate chain up to the trusted endorsers (e.g. the TPM manufacturer, or AMD for SNP). Then, the verifier checks the signatures of the metadata containing the reference values and the certificate chains up to the CAs it trusts. If both verifications succeed, the verifier compares the measurements to the metadata to assess whether the machine is in a good state, i.e., only trustworthy software is running on the machine.

The verifier can then extend this basic remote attestation with flexible and fine-grained policies suitable for their use case.

An *Attestation Report* contains:

- proof of the device identity
- measurements of the currently running software stack
- metadata for validation of the expected measurements
- configuration data and attributes for this device
- all certificates and certificate chains that have been used to sign the information above

The entire attestation report is shown in Figure 2. The colors of the different components in the attestation report indicate the various signers of those components. Furthermore, the complete report is signed to prove that it was generated on the proving platform.

We describe the elements of the attestation report as shown in Figure 2 in the following sections: We introduce the underlying PKI and keys in Section 4.1. Then, we describe the *Measurements* in Section 4.2 and the *Manifests* and *Descriptions* providing the reference values in Section 4.3.

4.1 Identity and PKI

To enable identification and authentication of devices, the attestation framework provisions devices with at least one asymmetric key-pair: the `Attestation Key (AK)`. The private portion of the `AK` must be generated by and never leave the trusted attestation hardware. This key-pair is used for signing the measurements (e.g., the TPM quote or SNP attestation report).

This key-pair can in theory be used for establishing secure channels. Depending on the used attestation technology, however, this might not be possible. E.g., the AMD SNP VCEK private key only signs SNP attestation reports, but not arbitrary data, which is required for establishing a secure channel. Therefore, more key-pairs can be generated for establishing secure channels and serving as the identity of the device. We recommend that these keys are generated in hardware as well, although it is not strictly necessary as the unique channel binding (see Section 4.6) prevents Man-in-the-Middle (MitM) attacks even in case of compromised keys, as discussed in Section 6.

As our attestation approach uses signatures for all reference values, we require a PKI for establishing trust in the public keys used to sign the data structures. Therefore, the described key pairs are integrated into a PKI with one or multiple root CAs, that confirm an entity's ownership of a key by issuing a certificate for the corresponding public key. The CAs must support revocation mechanisms, e.g., in case the respective private key was leaked.

The overall PKI framework is shown in Figure 3. Multiple root and intermediate CAs can be used to provide certificates for key pairs signing different metadata (*Entity 1-n* in Figure 3). Depending on the use case, these signing keys may belong to developers signing their (sub-)components and/or evaluators performing an additional evaluation and signing only software fulfilling previously defined criteria. In any case, the signers need to examine the software component and check for vulnerabilities before signing.

Additionally, the `AK` must be embedded into a trusted PKI. As shown in Figure 3, depending on the utilized attestation technology, the `AK` might already be embedded into a PKI offered by the manufacturer of the hardware trust anchor, which may be used for remote attestation as well (*Trust Anchor CA* in Figure 3). If this is not the case, the device must create Certificate Signing Requests

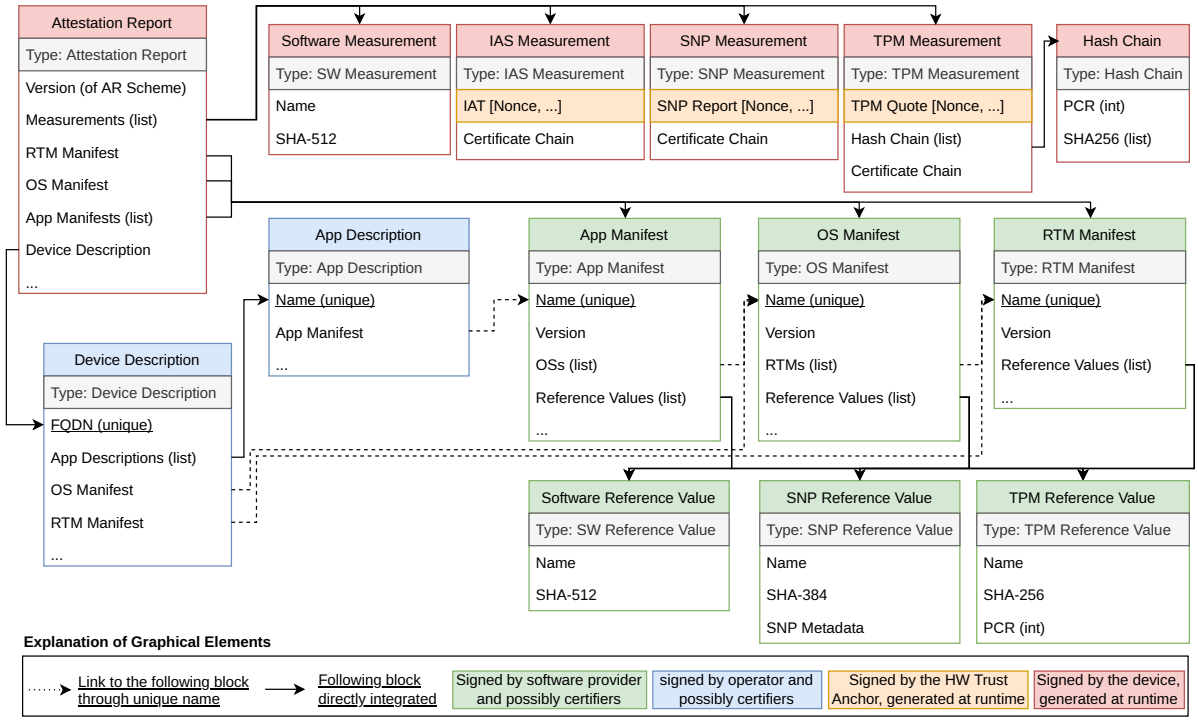

Figure 2: Attestation report

(CSRs) for the keys and send them to a CA (*Intermediate Ca* or *Root CA* in Figure 3), which evaluates that the keys fulfill the required properties before issuing certificates. The specific process is dependent on the used technology and described in Section 5.

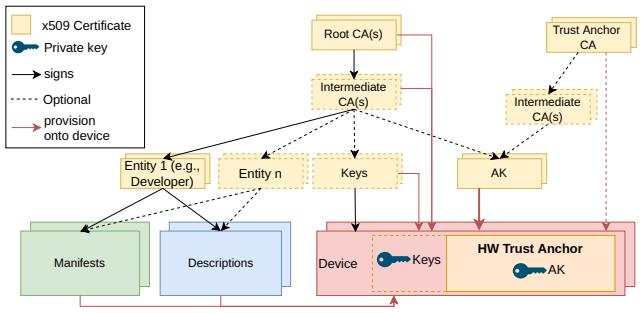

Figure 3: Overall PKI framework

4.2 Measurements And Chain of Trust

Our generic attestation framework requires the attestation evidence to be packed into a generic data structure called *Measurement*, so that platforms utilizing different attestation technologies can mutually attest each other. These objects are marked red in Figure 2. Each *Measurement* requires a *Type* field indicating the utilized attestation technology, such as TPM, AMD SNP or ARM IAS. Usually, only one of the measurement types is present, though multiple technologies can be used in parallel (e.g., a virtual TPM within an SNP VM). At the core of the *Measurement* data structure is the attestation evidence itself (marked orange in Figure 2), which is signed by the AK. The attestation evidence format depends on the utilized attestation technology, such as a TPM *Quote* or an *SNP Attestation Report*. This format cannot be transformed into a generic format as it is signed by the attestation hardware and therefore cannot be altered for signature verification. Furthermore, the *Measurement* contains said signature, the set of certificate chains of the signing key(s) and optionally further metadata required to interpret the attestation evidence. Additional metadata can e.g., be a list of hashes (*Hash Chain* in Figure 2), representing the order of measurements, in case it is required to recalculate the attestation evidence based on the reference values (see Section 5.2 for implementation specific details). The *Measurements* can be extended with further types to support future attestation technologies. Note that the *Attestation Report* and our attestation framework only ensure the interoperability between different attestation technologies and the interpretability of attestation evidence on the verifier side. The security of used software components must be ensured when evaluating and certifying *Manifests* and *Reference Values* as mentioned in Section 4.1, or during the extended policy verification as described in Section 4.5. We present exemplary software stacks for different device classes ensuring the measurement of all components in Section 5. Furthermore, it must be ensured, that only *Device Descriptions* are signed, that contain a *Manifest* for the attestation daemon itself.

4.3 Metadata

The measurements described above build the fundamental basis for remote attestation. However, making claims about the trustworthiness of the platform requires corresponding reference values, i.e., hashes, and metadata about the software. This metadata is provided in form of signed *Manifests*, marked in green in Figure 2. The metadata comprises a version number, a unique name, a list of other compatible *Manifests* and a list of software artifact *Reference Values*. These are the minimally required attributes for a basic attestation validation. The version number is required to distinguish different versions of the same software and enable the rejection of outdated versions. The unique name is required for linking different types of *Manifests* together. The list of compatible manifests is required as not every manifest might be compatible with every other manifest and the list of *Reference Values* represent the software running on the respective boot stage represented by the manifest.

To support flexibility, *Manifests* can be extended with further metadata to support different use-cases, such as a description of the *Manifest*, the name of the developer, or the certification level of an application.

We define three different types of *Manifests* to support re-usability of software artifacts and to take into account that software of differing boot stages is changed in different intervals (e.g., UEFI firmware usually is not updated as often as user space tools). All measurements for utilized artifacts are mutually exclusively mapped to these three types of *Manifests*. Each device requires one *Root of Trust for Measurements (RTM) Manifest* defining the initial starting point with the first, immutable software component and components of the early boot stages. Additionally, each device has one *Operating System (OS) Manifest* for the OS which takes control after the boot process and supports the execution of different isolated applications. Running on top of the OS, a device can run an arbitrary number of host and containerized applications described by *App Manifests*. As multiple instances of one application might run with different configurations, the *Attestation Report* contains an *App Description* per app instance. This in turn has a reference to the unique name of the *App Manifest* on which the instance is based. To avoid security issues arising from unsuitable combinations of applications, OSes, and RTMs, each *App Manifest* lists all compatible *OS Manifests*, and each *OS Manifest* lists the approved *RTM Manifests*, as represented by the dashed lines in Figure 2. Furthermore, all *Manifests* are linked together within the *Device Description* of the running platform, which is embedded into the *Attestation Report*. The *Device Description* furthermore contains the Fully Qualified Domain Name (FQDN) used to identify the platform.

Since software typically gets updated regularly, *Software Manifests* have a limited validity period. This validity period can be kept short (e.g., 1 day) so that the manifest validity alone is sufficient as a proof of up-to-dateness. In case of bugs, no new versions of the manifest would be signed to prevent usage of vulnerable software. However, if the validity period is longer (e.g., 1 year), there must be a revocation mechanism for invalidating manifests for software that is outdated or vulnerable to attacks. Possible revocation mechanisms can be realized based on existing revocation mechanisms for certificates, and range from revocation lists or querying an Online Manifest Status Provider (OMSP) (comparable to an Online Certificate Status Protocol (OCSP) for certificates) on the verifier side to providing up-to-date revocation information from the prover side in the *Attestation Report* in form of a *OMSP Responses* (OMSP Stapling).

4.4 Basic Validation

The *Attestation Report* is signed by the prover and transferred to the verifier for validation. The verifier is responsible for validating:

- the freshness of the *Attestation Report* with the used nonce
- the device identity, i.e., signature of the *Attestation Report*
- the integrity of the software stack by matching the *Measurements* with the *Reference Values* from the *Manifests*
- the correctness of all metadata structures based on their validity, revocation status, and signatures
- all used certificates by checking their certificate chain back to the list of trusted root CAs, validity and revocation status
- the mapping from *Device Description* to the *Manifests*, as well as between the different types of *Manifests*

After completing these steps, the verifier has validated basic information about the proving device and its software stack, which can be used for assessing its trustworthiness. In particular, it was proven that signed software is running on the device and that each artifact was signed by entities that are part of the trusted PKI. For mutual trust, each of the communication devices needs to act as both prover and verifier during remote attestation.

4.5 Policy-based Extended Validation

Knowing that only certain signed software is running on a platform might not be sufficient depending on the use-case. For example, one might want to restrict the software to certain versions, to allowlist or blocklist specific software, mandate the use of certain cryptographic algorithms, or require objects to be signed by multiple, different entities within different CAs. Furthermore, one might want to ensure that certain software, which is, e.g., responsible for measuring the components of an upper layer is present, to ensure that all software on the platform is actually measured.

As those additional policies heavily depend on the use-case, we design an interface for interacting with generic policy engines, which can be used to enforce policies based on the *Attestation Report* and its metadata. The basic validation outputs an *Attestation Result*, containing all relevant data, metadata and results from the basic validation. The metadata provided in the *Manifests* and *Descriptions* may contain additional use-case-specific attributes, which can also be examined during policy validation. This two-step validation process is shown in Figure 4. The verifier takes the *Attestation Report* as input and creates the *Attestation Result*. The policies engine uses this result together with user-specified policies to validate the report further. The final output is a boolean, signaling if the remote attestation was successful or not.

4.6 Attested Channel Establishment

For ensuring secure communication, it is not sufficient to only attest the trustworthiness of the software stack. We also need to bind the attestation evidence to the secure channel end-point used to communicate with the attested device. For this, we design *aTLS*, a

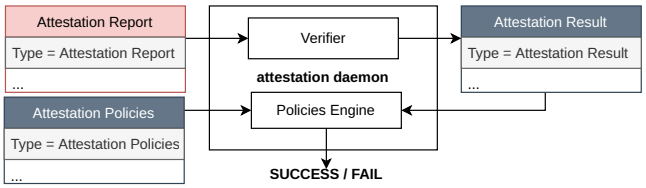

Figure 4: Two-step remote attestation with custom policies

Transport Layer Security (TLS)-based trusted channel protocol. We utilize TLS 1.3 [35] as the underlying protocol for establishing attested channels between devices. To be able to use common TLS 1.3 libraries and omit any modification to the libraries or protocol, we perform a post-handshake explicit attestation with unique channel bindings as defined by [32, 41]. We enable both server-side as well as mutual authentication and attestation.

For the unique channel bindings, we utilize the TLS 1.3 keying material exporter [35] with the exporter master secret, the label "EXPORTER-Channel-Binding" and a zero-length string for the context value as defined by [40]. We output keying material with a length of 32 bytes as defined by [40]. After establishing the TLS 1.3 channel, each attesting side generates its own attestation report with its own channel bindings as described, and uses these channel bindings as the nonce for the attestation report.

Both sides then send their attestation reports. During verification, both sides additionally verify that the received attestation report nonce matches the own calculated channel bindings.

5 IMPLEMENTATION

Our goal is to demonstrate the feasibility of our solution for different device classes and attestation technologies. Therefore, we implemented and evaluated PoCs on 4 exemplary platforms, from resource-constrained embedded IoT devices to powerful servers, with 4 different attestation technologies. We describe the utilized hardware platforms and attestation technologies in the hardware-specific paragraphs in the remainder of this section.

5.1 Hardware-agnostic Implementation

We implemented a *Remote Attestation Daemon (rad)*[2] in *Go* (golang), which can act as both prover and verifier. The *rad* can be run as a monolithic binary, but also exposes a gRPC[3], a Constrained Application Protocol (CoAP) [37], and a socket Application Programming Interface (API), enabling attestation also from services with restricted access to hardware, such as from within containers.

The *rad* can process attestation reports with Java Script Object Notation (JSON) [13] as well as with Concise Binary Object Representation (CBOR) [12] serialization. For JSON, we utilize JSON Web Signatures (JWS) [25] for signing and verifying the metadata objects. For CBOR, we use CBOR Object Signing and Encryption (COSE) [14].

Furthermore, we implemented *aTLS* as a *Go* trusted channel library on top of TLS as described in Section 4.6, providing a similar interface as the used underlying TLS library [6].

[2]https://github.com/Fraunhofer-AISEC/cmc
[3]https://grpc.io/

For provisioning, we built an Enrollment over Secure Transport (EST)-based provisioning server with additional hardware specific functionality, i.e., supporting TPM *Credential Activation* for the enrollment of certificates.

The remainder of this section describes the hardware-specific implementation for the different trust anchors.

5.2 TPM-specific Implementation

We implemented the DRTM TPM-based attestation platform on an Intel NUC11TNHv7 with an Intel Core i7-1185G7 CPU and an Infineon SLB 9670 TPM2.0. The NUC supports the Intel Trusted Execution Technology (TXT) technology for DRTM measured boot. We implemented the SRTM TPM-based attestation platform on an Intel NUC7PJYH with an Intel firmware TPM. The software stack comprises Intel firmware, grub bootloader, Ubuntu server 22.04 with a Linux kernel version 5.15 and user space applications. We implemented measured boot and remote attestation for both the SRTM and the DRTM approach. For DRTM, we use the Intel TXT technology.

We utilized a TPM supporting specification 2.0 [38] as the hardware trust anchor responsible for generating and storing private keys, storing measurements of software artifacts running on the platform, and attesting the current state of the platform.

5.2.1 Provisioning and Identity. During device provisioning, the *rad* creates an `AK` in the storage hierarchy as a *Fixed Parent*, *Fixed TPM*, *Restricted* primary key and performs *Credential Activation* with the EK. Furthermore, our CA validates the EK by following the certificate chain up to the TPM manufacturer CA.

Furthermore, the *rad* creates separate keys for establishing connections and signing *Attestation Reports*. To further strengthen the security, it binds those keys to the `AK` by certifying them with the `AK`. For this, the CA verifies that those keys were signed by a valid `AK` and also issue an x509 certificates for the keys. Furthermore, we provide the possibility to use software-based keys for establishing the TLS connection for increased performance.

5.2.2 Measurements and Chain of Trust. For remote attestation, all parts of the TCB must be verified. We utilized the TPM to store measurements of all TCB components. The device performs a measured boot utilizing the SRTM or DRTM approach as described in Section 2.1.

For the SRTM approach, the static PCRs are used. The PCR usage depends on whether a bootloader or the firmware boots the kernel (e.g., as an efi-stub). Without bootloader, the kernel is measured into PCR4 and PCR5. With bootloader, these PCRs record the bootloader, and the kernel is recorded in PCR8 and PCR9. The DRTM platform, uses the dynamic PCRs 17 and 18.

For measuring user space applications, we implemented two approaches: For the first approach, we utilized a minimal OS with few applications running in the root namespace and the actual workload of the device being processed in containerized applications. We packed and shipped all applications in the initramfs. We configured the kernel to append the initramfs to the kernel image as part of the *OS Manifest*. A read-only rootfs can be integrated through utilizing `dm-verity` for ensuring integrity of the file system during runtime. The root hash of the rootfs is part of the kernel

commandline and thus measured. For the second approach, we utilized the Integrity Measurement Architecture (IMA) for measuring user space applications in a more flexible setup.

5.2.3 Validation. We utilized a TPM *Quote* over the respective PCRs to attest the current state of the software stack.

The TPM-specific validation comprises two steps: Validating the signature of the quote with the AK certificate chain and comparing the content of the PCRs to the expected *Reference Values* as defined in the *Manifests*.

During validation, the reference values of the *Manifests* are compared to the measurements. The verifier checks that exactly the expected software is running on the platform by recalculating the values of the single PCRs and then aggregating those values to match the aggregated value stored in the TPM Quote.

5.3 AMD SNP-specific Implementation

We implemented the AMD SNP-based attestation platform on an AMD EPYC server with an AMD EPYC 7763 CPU. The host software stack comprises the firmware and Debian 11 with a custom Linux 5.19-rc6 kernel. The server initializes SNP VMs which run the actual workloads. These VMs run the *rad* to perform remote attestation.

5.3.1 Provisioning and Identity. We utilized the SNP VCEK as the AK. This key is embedded into a certificate chain up to the AMD SNP root CA and thus does not require signing by any custom CA. Further keys, e.g., for connection establishment, are generated during the first boot of a new VM using random data.

5.3.2 Measurements and Chain of Trust. AMD SNP measures the launch state of a VM before it is run. In simplified terms, the measurement in the SNP attestation report is the result of a hash chain comprising the hashes of the pages of the initialized memory of the VM. In our setup, this includes the UEFI and an additional page with hashes of the Linux kernel, initrd and kernel cmdline. The UEFI loads the kernel, the initrd and cmdline into the VMs memory and compares their hashes with the hashes in this page. This way, the hashes of the Linux kernel, the initrd and the cmdline are part of the launch measurement of the VM [31].

5.3.3 Validation. The verifier decodes the AMD SNP measurement, i.e., the SNP attestation report. It verifies the nonce against the supplied nonce. It verifies the attestation report signature utilizing the VCEK certificate, the VCEK certificate chain up to the AMD root CA and the VCEK certificate extensions against the report. The latter ensures that an AMD SNP TCB version cannot fake being a newer version. If this step succeeded, the data of the attestation report itself is verified. This includes comparing the measurements and reference values for the launch measure (digest) as well as expected and reported SNP report version, and SNP policy. The policy includes the Application Binary Interface (ABI) version, which is checked to be at least the version demanded by the *Manifest*, and several flags, such as debug, migration, or multi-threading flags, which must match the demanded values reference values. Furthermore, the minimum versions of the firmware and the TCB are verified. The TCB comprises the microcode, the bootloader, the Platform Security Processor (PSP) OS (TEE) version, and the SNP firmware version.

5.4 ARM PSA EAT-specific Implementation

We implemented the ARM PSA attestation token-based prover on an nRF5340 ARM Cortex-M33 Microcontroller Unit (MCU) with 1 MB of flash and 512 kB RAM. The software stack comprises the nRF Secure Immutable Bootloader (NSIB) as secure loader, MCUBoot as the bootloader, ARM Trusted Firmware-M as secure runtime firmware and Zephyr OS as the non-secure OS. We built a stripped, embedded version of the *rad* with a CoAP API in C. We utilized the Trusted Firmware-M IAS for generating the attestation tokens. As can be seen in Section 6, the overall size of an Entity Attestation Token (EAT) attestation report including all metadata and certificates is around 7 kB. This makes it feasible, to provision the metadata even onto tiny IoT devices. Currently, we implement the attestation framework but not the attested channel for Cortex-M platforms.

5.4.1 Provisioning and Identity. The nRF5340 implements the ARM TrustZone CryptoCell 312 as well as a Key Management Unit (KMU). This allows to provision a hardware protected key-pair onto the device to be used as the AK.

5.4.2 Measurements and Chain of Trust. The chain of trust starts with the inherently trusted NSIB. The NSIB validates the image of MCUBoot acting as the second stage bootloader. It implements a secure boot mechanism, i.e., the execution will be stopped if the MCUBoot image validation fails. MCUBoot measures both the secure world as well as the non-secure world images. It passes the measurements to the Trusted Firmware-M via a shared memory area if the validation of the image was successful. After Trusted Firmware-M completed its initial services, it switches to the normal world and passes control to the non-secure Zephyr OS. The normal world *rad* can then request AK-signed attestation tokens containing the measurements via the secure IAS service.

5.4.3 Validation. The *rad* validates the attestation report signature and certificate chain and the *Manifests* including the reference values. For the reference values, we used the type *Software Reference Value* as shown in Figure 2. The verifier examines the PSA attestation token: It validates the signature and certificate chain of the token. It compares the nonce to the nonce it provided. Finally, it compares the measurements of the non-secure and secure software to the reference values.

6 EVALUATION

In this section, we analyze our PoCs regarding performance and data overhead, and conduct a conceptual security evaluation.

6.1 Performance Evaluation

We utilize TLS 1.3 for establishing secure channels between devices. As we utilize a post-handshake attestation, we add an additional Round-Trip Time (RTT) and additional exchanged attestation data to the handshake. We therefore compare our solution to a classic TLS 1.3 handshake and connection regarding data and performance overhead.

6.1.1 Performance Overhead. For measuring the handshake duration, we performed 100 handshakes with mutual TLS. We implemented a default golang TLS client and server application as reference, run it on two NUC7PJYH and compared it with aTLS run

on the SRTM and SNP platform. We measured the times via the `time` package in go. We performed a client-side measurement with the total time as shown in Table 1 as the time between the start of the handshake and the sending of the first transport message by the client.

Table 1: Handshake duration

	TLS 1.3	aTLS (SNP)	aTLS (TPM)
Total ms	2.456	31.2	476.1

The time for SNP was measured between two VMs running on the same host i.e., both attestation reports were fetched sequentially. If SNP VMs on different hosts perform mutual attestation, we expect a speed-up of around 8 ms, as the attestation reports can then be fetched in parallel. The time to establish a connection with the TPM is significant. However, the TPM requires already 429 ms of the 476 ms total handshake time to fetch a quote. This is a general TPM problem. Note that the time heavily depends on the utilized TPM. For the test, we utilized the Intel firmware-TPM of the NUC7PJYH. We did not measure the ARM PSA attestation token handshake duration, as we did not implement TLS on the embedded device.

6.1.2 Data Overhead. Compared to a default TLS 1.3 handshake, both sides have to transfer their attestation reports to establish an attestedTLS connection. As the data exchanged during a default TLS 1.3 handshake varies heavily depending on the used cipher suites and certificate chains, we only measure the size of the exchanged attestation reports for our exemplary platforms. The results can be seen in Table 2. For a mutual attestation, the overall overhead is twice the size of the attestation report plus the overhead for wrapping it into TLS record protocol packets. The columns marked with an asterisk (*) represent the size of the gzip-compressed attestation reports to reduce the size of the additionally transferred TLS-data.

Table 2: Attestation report size on the exemplary platforms

	TPM	TPM*	SNP	SNP*	IAS
CBOR (bytes)	15004	6235	17786	6553	7313
JSON (bytes)	90490	38318	43131	22824	-

The reason, the CBOR report for SNP is larger than for TPM, while the JSON report for SNP is smaller than for TPM, is because digests are serialized as hexadecimal strings in JSON, while stored in binary in CBOR. This increases the size of the JSON TPM attestation report, as it contains more reference values compared to the SNP attestation report, and each digest requires twice the space in JSON compared to the binary representation in CBOR.

6.2 Security Evaluation

We evaluate the security of our solution based on the threat model and security requirements as defined in Section 3.

REQ-I (Secure Channel) We rely on the security of TLS 1.3 for ensuring authentication, confidentiality, freshness, and Perfect Forward Secrecy (PFS), as we do not modify the handshake and connection establishment at all. Instead, we perform a post-handshake attestation utilizing the established secure channel. Therefore, an attacker would need to compromise the security of TLS 1.3 to break the secure channel. We perform unique channel binding by using the TLS 1.3 master exporter secret as defined by [40]. The attestation is therefore bound to the long-term static identity and the secure channel instance, i.e., the session itself. This prevents MitM and relay attacks even in case the long-term TLS identity gets compromised. In such an attack, an attacker could provision keys onto a rogue device M and establish a TLS connection to a device A. At the same time, the attacker establishes a connection to a good device B and retrieves B's attestation report. It then forwards this attestation report to A. However, as B embeds its own channel bindings resulting from the connection with M into its attestation report, A would notice the relay attack and refuse the connection.

REQ-II (Trustworthiness) An attacker may use one of the following two approaches to manipulate the representation of the software stack: a) Manipulate the measurements to deviate from the actual software state, e.g., by executing software that is not measured, and b) Include the hash of the malicious software in the measurements and manipulate the provided metadata to make it look as if this software was expected.

For a), the attacker is limited by the use of the implicitly trusted first boot component (e.g., CRTM) which ensures that at least the measurement of the first mutable component in the chain of trust is stored in the hardware trust anchor. Additionally, the attacker can not get any of the intended software to load any component without measuring it, since intended software components are only signed if they ensure the integrity and authenticity of all following software components.

Thus, an attacker is limited to attack scenario b). While a Dolev-Yao attacker would be prevented from manipulating the *Attestation Report*, as they cannot forge the signature, an insider attacker may be capable of requesting a signature for an attestation report with customized metadata files. The measurements from the hardware trust cannot be manipulated and include the hash of the malicious software, so the attacker needs to generate *Software Manifests* claiming these measurements belong to trustworthy software. However, the *Manifests* used for describing the expected software need to be signed by trusted entities leaving the attacker incapable of providing properly signed *Manifests* and allowing the verifier to detect the mismatch between measurements and expected values (REQ-IIa).

An internal attacker may run a downgrade attack and (try to) cause the machine to execute an outdated version of a software component with known vulnerabilities for which a signed *Software Manifest* exists. However, we defined in the concept that, in case of new versions and/or known vulnerabilities, either no new manifests would be issued in case of short validity terms, or, in case of longer validity terms, a suitable revocation mechanism, e.g., OMSP responses in the *Attestation Report*, would allow a verifier to notice the presence of outdated software components (REQ-IIb).

REQ-III (Channel Binding) We perform a post-handshake explicit unique channel binding. This approach binds the attestation

to the long-term device identity as well as to the secure channel uniquely in time, i.e., to the specific secure channel instance. This prevents relay attacks, where the attacker has control over a trustworthy and a compromised machine and extracts the keys for connection establishment from the trustworthy device to use it on the compromised machine as well. Such an attacker M would intercept the secure channel establishment from the legitimate Peer A to the legitimate Peer B under their control and itself establish secure channels to both peers. For the connection establishment to A, it would use the keys previously extracted from B. After the TLS handshake, M would receive the attestation report from B and forward it to A to impersonate a legitimate device. However, A would detect that the attestation report was not created by M, as B would use its own channel binding as the nonce during attestation report generation, which includes not only its long-term static key-pair, but also the freshly generated ephemeral key-pair used for the TLS handshake.

7 RELATED WORK

Remote attestation is a broad research area. This section will limit itself to giving an overview about attestation frameworks and remote attestation protocols i.e., the establishment of attested channels between devices.

The Trusted Computing Group (TCG) Trusted Network Communications (TNC) [22] is a client-server scheme for authenticating and attesting endpoints to servers guarding access to networks. The specification also uses a generic format for integrity information. The architecture is for TPM-based platforms only and differs from our solution, as we enable mutual attestation without involving a central party.

The *go-attestation* project [5, 18] provides a TPM-based device identity and attestation framework for known communication partners. We utilize parts of this library for the TPM-driver implementation. WS-attestation is a TPM-based attestation architecture for web services [42]. They aim to address granularity, e.g., when measuring text-based configuration files, as well as dynamic user-space components. In their attestation architecture, they make use of a third-party validation service for the attestation process. While this differs from our approach, parts of the architecture could be used to complement our TPM-based attestation mechanism. Additional frameworks [9, 34] exist, which also focus on TPM-based attestation only.

In [28], the authors propose a generic attestation framework for IoT-devices. It focuses on one-sided attestation of IoT devices and supports TPM and TEE-based attestation. A generic scheme for exchanging metadata and establishing trust in individual hashes is not considered. MAGE proposes a design for mutual attestation of groups of enclaves without trusted third parties [15]. It utilizes a group attestation scheme and implement it for Intel Software Guard Extensions (SGX) enclaves. These just presented frameworks target different TEE and CC-based technologies, but do not support a universal approach which also covers, e.g., TPMs.

Furthermore, there is hardware-independent research: [33] provides a detailed analysis on attestation of TEEs and CC technologies. They compare the enclave-based attestation frameworks Veracruz [16], Veraison [29], Open Enclave [8], Enarx [4], Oak [7] and AWS Nitro [36]. [30] compares the attestation mechanisms of Intel SGX, ARM TrustZone, AMD SEV and RISC-V. [24] proposes a flexible mechanism for remote attestation for different use-cases. The focus is on an attestation protocol language and the coordination of various roles such as provers, verifiers and relying parties. A generic attestation metadata format for the concrete implementation of hardware technologies is out of scope.

The RATS architecture [11] defines a generic approach for attestation procedures. However, as described in Section 1, with RATS, the burden of assessing the trustworthiness of provided measurements is placed on the verifier side whereas our paper makes the prover responsible for this task. Furthermore, our trust model differs from [11], which involves three parties in the attestation: The *Attester*, the *Verifier* and the *Relying Party*. This requires to also secure the potential second communication channel with a trust anchor between *Verifier* and *Relying Party*. [11] further states that for a stronger level of security, the *Relying Party* might perform some kind of attestation as well, to assess the trustworthiness of the *Verifier*. In contrary, our approach combines the verifier and relying party into one entity on the verifying platform and therefore requires only one communication channel.

In [32] and [39], surveys of attested channel protocols are conducted. Both publications analyze previous proposals and outline various weaknesses. They then propose improved solutions based on TLS [32] or a custom protocol [39]. [32] uses TLS as well, but utilizes an intra-handshake unique channel binding. Both protocols could be incorporated into our framework as variants for the secure channel establishment.

8 CONCLUSION

In this paper, we presented a hardware-agnostic attestation framework for different classes of devices. The attestation framework enables mutual validation of the trustworthiness of devices based on device identity and integrity of their software stacks.

Our approach can utilize different hardware mechanisms, requires no central trusted third party, and the verifier is not required to carry a huge allowlist of all possible valid software component measurements. Instead, trust is established through using cryptographic signatures and a PKIs so that the verifying party can perform the remote attestation solely based on one or multiple common Roots of Trust.

We showed the feasibility of our concept by implementing a PoC for three different hardware trust anchors: TPM, AMD SEV-SNP, as well as the TEE-based ARM PSA attestation token.

Additionally, we evaluated the security of our solution and showed that our concept enables establishing a mutually attested, secure communication channel in the face of our defined threat model.

ACKNOWLEDGMENTS

This work partially has been supported by the German Federal Ministry of Education and Research (BMBF) in the context of the InDaSpacePlus project (no. 01IS17031) as well as the VE-DIVA-IC project (no. 16ME0282).

REFERENCES

[1] 2018. *TrustZone® technology for Armv8-M Architecture.* Technical Report. ARM.

[2] 2020. *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More.* White Paper. AMD.

[3] 2021. *SEV Secure Nested Paging Firmware ABI Specification.* Technical Report # 56860. AMD.

[4] 2023. Enarx. Retrieved March 15, 2023 from https://enarx.dev/docs/Technical/Introduction

[5] 2023. go-attestation. Retrieved March 15, 2023 from https://github.com/google/go-attestation

[6] 2023. Go crpyto/tls package. Retrieved March 15, 2023 from https://pkg.go.dev/crypto/tls

[7] 2023. Oak. Retrieved March 15, 2023 from https://github.com/project-oak/oak

[8] 2023. Open Enclave SDK. Retrieved March 15, 2023 from https://github.com/openenclave/openenclave

[9] Stefan Berger, Kenneth Goldman, Dimitrios Pendarakis, David Safford, Enriquillo Valdez, and Mimi Zohar. 2015. Scalable Attestation: A Step Toward Secure and Trusted Clouds. In *2015 IEEE International Conference on Cloud Engineering*. 185–194. https://doi.org/10.1109/IC2E.2015.32

[10] Henk Birkholz, Thomas Fossati, Yogesh Deshpande, Ned Smith, and Wei Pan. 2022. *Concise Reference Integrity Manifest.* Internet-Draft draft-ietf-rats-corim-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-rats-corim/00/ Work in Progress.

[11] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. 2023. Remote ATtestation procedureS (RATS) Architecture. RFC 9334. https://doi.org/10.17487/RFC9334

[12] C Bormann and P Hoffman. 2020. RFC 8949 Concise Binary Object Representation (CBOR). (2020).

[13] Tim Bray. 2014. *The javascript object notation (json) data interchange format.* Technical Report.

[14] August Cellars. 2022. RFC 9053 CBOR Object Signing and Encryption (COSE): Initial Algorithms. (2022).

[15] Guoxing Chen and Yinqian Zhang. 2022. MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 4095–4110. https://www.usenix.org/conference/usenixsecurity22/presentation/chen-guoxing

[16] Arm Ltd. Confidential Computing Consortium. 2021. eracruz: privacy-preserving collaborative compute. https://github.com/veracruz-project/veracruz

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

[18] Matthew Garrett and Tom DNetto. 2019. Using TPMs to cryptographically verify devices at scale. Open Source Summit, North America 2019.

[19] Gartner. 2020. https://www.gartner.com/en/doc/iot-security-primer-challenges-and-emerging-practices

[20] Gartner. 2022. https://www.gartner.com/en/newsroom/press-releases/2022-02-09-gartner-says-more-than-half-of-enterprise-it-spending

[21] Trusted Computing Group. 2013. TCG D-RTM Architecture.

[22] Trusted Computing Group. 2017. TCG Trusted Network Communications TNC Architecture for Interoperability Version 2.0 Revision 13.

[23] Trusted Computing Group. 2021. TCG PC Client Platform Firmware Profile Specification Version 1.05 Revision 23.

[24] Sarah C. Helble, Ian D. Kretz, Peter A. Loscocco, John D. Ramsdell, Paul D. Rowe, and Perry Alexander. 2021. Flexible Mechanisms for Remote Attestation. *ACM Trans. Priv. Secur.* 24, 4, Article 29 (sep 2021), 23 pages. https://doi.org/10.1145/3470535

[25] Michael Jones, John Bradley, and Nat Sakimura. 2015. JSON web signature (JWS). *Internet Requests for Comments, RFC* 7515 (2015).

[26] David Kaplan, Jeremy Powell, and Tom Woller. 2016. *AMD Memory Encryption.* White Paper. AMD.

[27] David Kaplan, Jeremy Powell, and Tom Woller. 2020. *AMD SEV-SNP: Strengthening VM Isolationwith Integrity Protection and More.* Technical Report. Technical Report. Advanced Micro Devices Inc.

[28] Kyeong Tae Kim, Jae Deok Lim, and Jeong-Nyeo Kim. 2022. An IoT Device-trusted Remote Attestation Framework. In *2022 24th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 218–223.

[29] Arm Ltd. 2021. Attestation verification service / veraison. https://github.com/veraison

[30] Jämes Ménétrey, Christian Göttel, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2022. An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments. *arXiv preprint arXiv:2204.06790* (2022).

[31] Dov Murik. 2022. [RFC PATCH] i386/sev: Support measured direct -kernel boot on SNP. https://lore.kernel.org/qemu-devel/20220329064038.96006-1-dovmurik%40linux.ibm.com/#r

[32] Arto Niemi, Vasile Adrian Bogdan Pop, and Jan-Erik Ekberg. 2021. Trusted Sockets Layer: A TLS 1.3 based trusted channel protocol. In *Nordic Conference on Secure IT Systems*. Springer, 175–191.

[33] Arto Niemi, Sampo Sovio, and Jan-Erik Ekberg. 2022. Towards Interoperable Enclave Attestation: Learnings from Decades of Academic Work. In *2022 31st Conference of Open Innovations Association (FRUCT)*. 189–200. https://doi.org/10.23919/FRUCT54823.2022.9770907

[34] Wojciech Ozga, Do Le Quoc, and Christof Fetzer. 2021. TRIGLAV: Remote Attestation of the Virtual Machine's Runtime Integrity in Public Clouds. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. 1–12. https://doi.org/10.1109/CLOUD53861.2021.00013

[35] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[36] Amazon Web Services. 2023. Amazon AWS Nitro. Retrieved March 15, 2023 from https://aws.amazon.com/ec2/nitro/

[37] Z Shelby, K Hartke, and C Bormann. 2016. RFC 7252: The Constrained Application Protocol (CoAP), Internet Engineering Task Force (IETF), 2014. *Acessado em* 10, 10 (2016), 2017.

[38] Trusted Computing Group (TCG). 2019. Trusted Platform Module Library Specification, Family 2.0, Revision 01.59.

[39] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. 2020. Establishing secure communication channels using remote attestation with TPM 2.0. In *International Workshop on Security and Trust Management*. Springer, 73–89.

[40] S Whited. 2022. RFC 9266: Channel Bindings for TLS 1.3.

[41] Nicolas Williams. 2007. *RFC5056: On the use of channel bindings to secure channels.* Technical Report.

[42] Sachiko Yoshihama, Tim Ebringer, Megumi Nakamura, Seiji Munetoh, and Hiroshi Maruyama. 2005. WS-Attestation: Efficient and fine-grained remote attestation on web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE.