Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare

Project Acronym: **ASCLEPIOS**
Project Contract Number: **826093**

Programme**: Health, demographic change and wellbeing**
Call: **Trusted digital solutions and Cybersecurity in Health and Care to protect privacy/data/infrastructures**
Call Identifier: **H2020-SC1-FA-DTS-2018-2020**

Focus Area: **Boosting the effectiveness of the Security Union**
Topic**: Toolkit for assessing and reducing cyber risks in hospitals and care centres**
Topic Identifier: **H2020-SC1-U-TDS-02-2018**

Funding Scheme: **Research and Innovation Action**

Start date of project: 01/12/2018 Duration: 36 months

Deliverable:

D4.2 Remote attestation of workloads in ITEEs

Due date of deliverable: 31/05/2020 Actual submission date: 30/05/2020

WPL: Nicolae Paladi/RISE

Dissemination Level: Public

Version: 0.9

Table of Contents

List of Figures and Tables

Figures

Tables

Status, Change History and Glossary

Status:	Name:	Date:	Signature:
Draft:	Arash Vahidi Nicolae Paladi	2020-02-27	N/A
Reviewed:	Giannis Ledakis	2020-05-22	N/A
Approved:	Tamas Kiss	2020-05-30	N/A

Table 1: Status Change History

Version	Date	Pages	Author	Modification
v0.1	2020-01-07	8	Nicolae Paladi	Create working document
V0.2	2020-04-xx	xx	Arash Vahidi	Updated first chapters
V0.3	2020-05-07	xx	Giannis Ledakis, Sofianna Menesidou (UBI)	Updates for TPM
V0.4	2020-05-xx	31	Christiaan Hillen	Added attestation in ASCLEPIOS
V0.5	2020-05-14	31	Arash Vahidi	Preparing for internal review
V0.6	2020-05-22	31	Giannis Ledakis	Changes during the review
V0.7	2020-05-22	31	Arash Vahidi	Updated after review from UBITECH
V0.8	2020-05-26	31	Arash Vahidi	Minor clean-ups before submission
V0.9	2020-05-29	31	Nicolae Paladi	Add section 1.3.2 outlook; extend Section 6 – summary.

Table 2: Deliverable Change History

Glossary

CPU	Central Processing Unit
ROM	Read-Only Memory
TA	Trusted Application
TAM	Trusted Application Manager
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
TEEP	Trusted Execution Environment Platform
RATS	Remote Attestation Procedures Architecture
EAT	Entity Attestation Token
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
TPM	Trusted Platform Module
SoC	System-on-Chip
CA	Certificate Authority
CPU	Central Processing Unit
DoS	Denial of Service
IETF	Internet Engineering Task Force
ITEE	Isolated Trusted Execution Environment
MEE	Memory Encryption Engine
OS	Operating System

Table 3: Glossary

1 Introduction

When software executes in an unknown or foreign environment such as a third-party cloud, some guarantees about correctness of execution may be needed. For example, the target environment may need to provide evidence that the software has really been executed and that the environment was structured and functioning as expected.

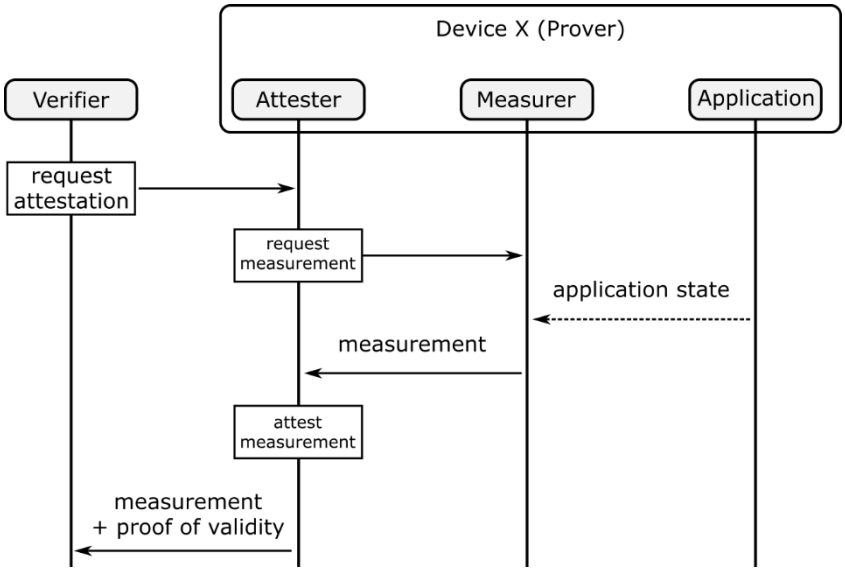

Figure 1, essential components of remote attestation.

Sometimes an *attestation* mechanism can be used to provide the required proof of validity. This proof may be limited to certain part of the system and/or require pre-existing trust in a local or remote component. In the ASCLEPIOS project we consider remote attestation of firmware and workloads in isolated Trusted Execution Environments (TEEs). This will provide the guarantees that about involved systems to remote users, which is crucial when operating on highly sensitive medical data.

1.1 Scope

This document introduces background and principles of remote attestation for Trusted Execution Environment (TEEs). The scope of this is limited to a selection of existing hardware and software technologies that can be used within the ASCLEPIOS project.

1.2 Objectives

The main objectives of this document are:

1. Provide the required background for understanding remote attestation
2. Describe the current state of the art in remote attestation
3. Provide a set of protocols for remote attestation within the ASCLEPIOS project

1.3 Relation to other work packages and deliverables

This deliverable is part of ASCLEPIOS WP4 (Isolated Execution and Medical Device Security). As such it is related to other deliverables in the work package as following:

1. *"D4.1 Design of protocols for key, firmware and workload management in ITEEs for medical devices"* considers protocols for management of TEEs in medical devices and cloud systems
2. *"D4.2 Remote attestation of workloads in ITEEs"* (this document) considers remote attestation of these components

3. *"D4.3 Interoperability of ITEEs in the context of eHealth systems"* considers interoperability between architecture and system-dependent mechanisms and solutions in D4.1 and D4.2

1.3.1 *Application in ASCLEPIOS*

Within the project this work aims to enable the security requirements and use cases highlighted in WP1 (Reference Architecture and Use Cases) and the architecture and protocols presented in WP2 (Operations on Encrypted Health Data and Privacy-Preserving Health Data-Driven Analytic) and WP3 (Access Policies and Enforcement Middleware). The result of this work will be used in later work packages, in particular WP5 (Platform Integration and Finalization), specially T5.2 and T5.3 where it will be implemented.

1.3.2 *Outlook*

The review of remote attestation procedures described in this document accompanies work on the **implementation** of the Trusted Execution Environment Platform (TEEP) architecture defined by the Internet Engineering Task Force (IETF) [28]. The TEEP Architecture (see Section 4.8) describes a platform-independent mechanism for deployment of workloads in Trusted Execution Environments (TEEs).

***TEEP Deployer* (TEEPD)** is an on-going **implementation of the TEEP Architecture** led by RISE within the ASCLEPIOS project. Implementation work on TEEPD continued throughout WP4 and was supported by the insights collected and reported in deliverables D4.1, D4.2 and D4.3. Implementation work on TEEPD will continue beyond the conclusion of WP4. **Throughout WP5**, TEEPD will be further refined and integrated with the ASCLEPIOS platform, while **in WP6** TEEPD will be evaluated as part of the demonstrator.

This document supports the *integrity attestation* functionality of TEEPD: a workload deployed in a TEE must go through an *integrity attestation* procedure in order to be trustworthy. Depending on the chosen TEE architecture, integrity attestation may include the workload itself, the TEE where the workload is running, or additional information about the underlying platform. This document describes several integrity attestation protocols supported by server platform vendors. TEEPD wll implement support for a suitable integrity attestation protocol, depending on the target hardware architecture chosen for the ASCLEPIOS framework in WP5.

2 Remote attestation: concepts and background

Remote attestation is an important security component in the ASCLEPIOS project. To describe what it does and how it is used we first need to explain some concepts and ideas behind it, starting with some historic information about computer security.

2.1 Trust

Attestation is mainly a component for establishing trust. As such it may be important to recapitulate some core ideas behind trust in computing devices.

2.1.1 *Trusted Computing Base*

The *Trusted Computing Base* (TCB) is generally considered to be the core software/firmware/hardware components that are critical to the system security [1]. In practice, this tends to cover core system components that have higher capabilities / privileges than normal components. Hence a security vulnerability in the TCB will have severe consequences.

In a secure system TCB boundaries must be well-defined and well-protected. It is therefore generally believed that a system with a large TCB is due to higher complexity and larger attack surface harder to secure than a one with a smaller TCB. Hence one core idea in computer security is to minimize the TCB size.

2.1.2 *Isolation*

Isolation provides separation between components which is a prerequisite for many security functions. For example, the TCB must protect itself from other software and therefore uses memory isolation mechanisms. Similarly, an operating system must isolate applications from each other and itself.

Isolation technologies are beyond the scope of this document but are discussed briefly in D4.1 and D4.3. For the remaining of this document we will assume sufficient isolation mechanisms are in place.

2.1.3 *Measurement*

A *Measurement* collects evidence from a component by direct observation. For example, an application may be measured by computing a cryptographic hash of its binary. A running application may be measured by computing a cryptographic hash of its memory contents.

A measurement can be *reported* to another entity, which often includes mechanisms to prove its authenticity. This is often done by some type of cryptographic signature.

2.1.4 *Root of Trust*

A *Trust Anchor* is an entity that we assume can be trusted explicitly. A *Root of Trust (RoT)* is a Trust Anchor which is the root of a trust chain. For example, a device may come with a cryptographic certificate which it inherently trusts and used to authenticate all other certificates it encounters. A RoT may be implemented in hardware, software or a combination of both.

A system may contain multiple Roots of Trust for different purposes, for example a RoT for Storage and another for Measurement. As a special case, the RoT for Measurement may be static (initiated at boot time) or dynamic (initiated at a later time).

2.2 Trusted Computing

Trusted Computing is a concept in computer security where the computer in question may not be under full control of a user who still wants to ensure its security [2]. Some important components in trusted computing are

- Secure boot: boot into a well-defined and trusted configuration
- Strong memory isolation: provide memory isolated from other components and interfaces

- Sealed storage: secure storage of data at rest
- Integrity measurements: mechanisms to validate state of the system
- Remote attestation: mechanisms to proof validity to a third party

Much of the recent work in bringing trusted computing to the masses can be contributed to the Trusted Computing Platform Alliance (TCPA) and later the Trusted Computing Group (TCG) [3]. Among other activities, the Trusted Computing Group published the Trusted Platform Module (TPM) specification (which later became ISO/IEC 11889) for a tamper-proof hardware that aims to improve platform security in a standard computer and enable trusted computing [4] [5]. The TPM specification covers a number of subjects but of special interest to us are the state management and the platform attestation functionality. We will discuss these briefly as they have some historical significance.

2.2.1 *Trusted Platform Module*

Trusted Platform Module (TPM) is an international standard for a secure cryptographic coprocessor, often as a dedicated security microcontroller [6]. The first major version of the standard defined TPM 1.2, followed by TPM 2.0 in the second revision [7].

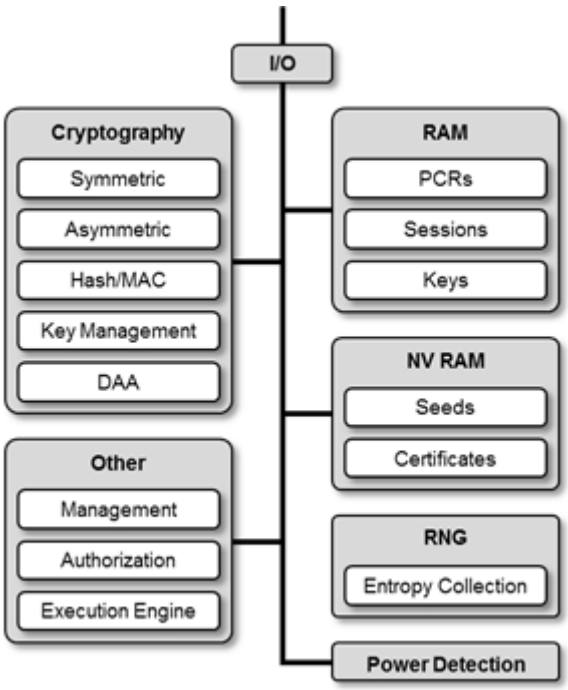

Figure 2, an overview of the TCG's TPM Architecture (image from [7]).

An overview of the internal architecture of the TPM is shown in Figure 2. The main components of include cryptographic operations, random number generation and non-volatile storage. The software and the API are specified by the TPM Software Stack (TSS).

While the TPM is not the main focus of this work, studying it will allow us to better understand more recent technologies such Intel SGX. However, due to space constraint in this work we will limit ourselves to functions for state measurement and attestation.

The TPM contains several 160-bit[1] Platform Configuration Registers (PCRs) for use with various TPM commands. Here we are mainly interested in the Reset and Extend commands:

RESET: PCR[n] = 0x00..0 (or 0xFF..F)

[1] Latest revisions support wider registers.

EXTEND: PCR[n] = Hash { PCR[n] || data }

Starting from a known PCR value (i.e. Reset), one can measure a series of components (e.g. BIOS, firmware, bootloader, OS, application, …) to extend registers. This creates a hash chain where the final value is a cryptographic evidence for the state of all these components.

As a PCR cannot be directly written to, one may trust the value provided by the TPM to represent what it claims. Unfortunately, a PCR can be freely read from which makes it unsuitable as an unforgeable proof of system state (e.g. no resistance to replay attacks). To address this the TPM can make provide a proof of validity by signing the current value (plus a nonce to hinder replay attacks and some additional data):

QUOTE: $S = Sign_{key}$ { PCR[n], nonce, …}

Assuming the private key in this operation is kept secret by the TPM, the signature can act as a secure proof of the validity. In other words, by using its secret key the TPM attests that the presented PCR value is real. This is essentially the TPM *Root of Trust for Reporting* (RTR).

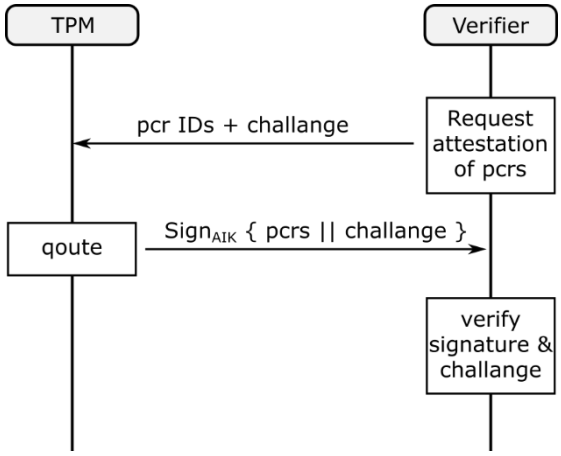

Figure 2, TPM quotation with AIK.

The TPM contains a unique secret Endorsement Key (EK), which in theory could be used for this signature. However, due to privacy implications often another key named AIK is used for this purpose. We will revisit this subject in Section 3.2.

2.3 Attestation

We have briefly looked into how attestation in TPM works. In the following we will revisit attestation, this time as a generic operation.

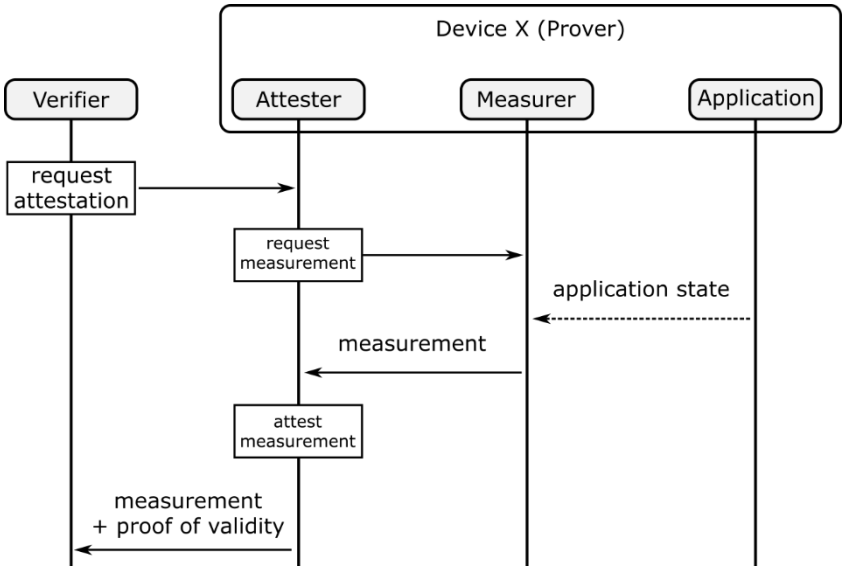

Figure 3, remote attestation revisited.

In attestation, a party often called the *Verifier* attempts to verify something provided by a yet untrusted *Prover*. As shown in Figure 3, the *Prover* may contain additional components used for attestation.

We consider two main types of attestation, local and remote:

1. **Local attestation**: the *Prover* wants to prove validity of something to a *Verifier* on the same machine. This requires existence of some authority on the device that both parties trust, which is normally the TCB.
2. **Remote attestation**: the *Prover* is located on a remote device, hence the *Verifier* also needs to establish trust with the remote TCB [8]. In particular, the *Verifier* must ensure that (1) the remote device is indeed the intended target, (2) that communication with the remote TCB is not susceptible to Man-in-the-Middle attacks (MITM), (3) that the remote TCB is in the intended state and (4) the remote TCB provides sufficient evidence of successful local attestation for the *Prover*.

Typically, the *Verifier* is trying to verify correctness of something static within the *Prover*, for example the authenticity and integrity of an *Application*. However, in some special cases the *Verifier* may instead try to verify the authenticity of a dynamic event, for example the result of executing the *Application* with some specific input.

3 State of the art

In this chapter we will consider a number of approaches to remote attestation including commercially available solutions and recent research efforts.

3.1 Requirements

In [8] the authors present five principles that are crucial for attestation:

1. Freshness: the measurement reflects the current state of the system
2. Comprehensive information: the full internal state is measured
3. Constrained disclosure: the Prover machine can decide how to respond to each Verifier
4. Semantic explicitness: The semantic content of attestations should be explicitly presented in logical form.
5. Trustworthy mechanism: provide evidence that the mechanisms attestation rely on are trustworthy

As shown momentarily, not all protocols and implementations adhere to all principles. For example, some protocols include only a limited part of the internal state, sometimes only a small section of the execution trace. Trustworthiness is another principle that is handled very differently.

In most cases however they follow a simple structure

1. An attestation request is received with accompanying nonce to ensure freshness
2. Something is evaluated (measured) and presented as a cryptographic digest
3. The digest and nonce are cryptographically signed and sent to the requester

For this model the threats can also be simplified to the following

1. The measurement may not reflect the actual state
2. The cryptographic signature (or whatever mechanisms is used) may have been forged

As demonstrated below, these will manifest differently in different protocols.

3.2 Privacy considerations

Recall from previous chapters that each TPM contains a unique secret Endorsement Key (EK). Using this key for signing attestation results has however some privacy (Verifiers can uniquely identify a Prover) and deniability (anyone can verify all claims) implications. To address this, normally an alias of the Endorsement Key named AIK (Attestation Identity Key) is used in TPMs with help of endorsement from a trusted third-party CA.

Use of the trusted third-party to validate reports may not always be optimal as it would (1) require high availability and (2) an insecure or even malicious third-party could cause significant damage. The Direct Anonymous Attestation scheme (DAA) aims to address this by using a zero-knowledge protocol [9]. An implementation of DAA using RSA has been adopted for use in TPM v1.2. Unfortunately, this implementation has its own set of privacy issues. Several attempts have been made to improve it, among others the Enhanced Privacy ID (EPID) [10]. EPID still requires a third-party (the Issuer) but takes steps to minimize the amount of damage a malicious or infected third-party can inflict[2].

Note that while it is useful to know and understand attestation privacy issues and how they are met, such issues are generally not applicable to ASCLEPIOS and its infrastructure. As such we will mainly ignore these issues in the remaining of this document.

[2] Note that the EPID scheme makes use of elliptic-curve cryptography, which is not supported by older TPM 1.2 devices. It is however supported by TPM 2.0 and Intel SGX, which we will look into later in this document.

3.3 Protocols and approaches

We consider a number of approaches to remote attestation, including some that are hardware independent, require some generic security hardware or require dedicated security hardware.

3.3.1 *Software-based attestation*

SWATT is a software-based approach to remote attestation for network sensor devices without any special hardware support [11]. A Verifier requests measurement of Provers memory in a pseudorandom order and knowing device characteristics (performance in particular) it can with certain probability detect an unauthentic response. Pioneer implements SWATT on a PC by first measuring the integrity of the Prover TCB with a time optimal checksum implementation and then measuring the application in question with a normal hash function [12]:

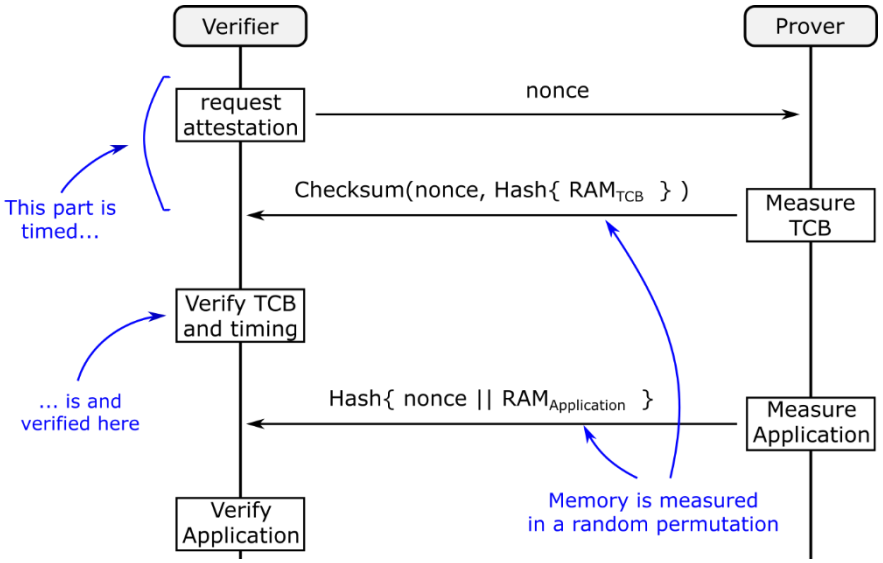

Figure 4, the Pioneer protocol.

The idea of using some indisputable code execution (ICE) was generalized in [13] and later improved to take into account caches in [14]. The general idea can be extended to different applications, for example in SAKE ICE is used to implicitly attest state of both ends in a key agreement protocol [15]:

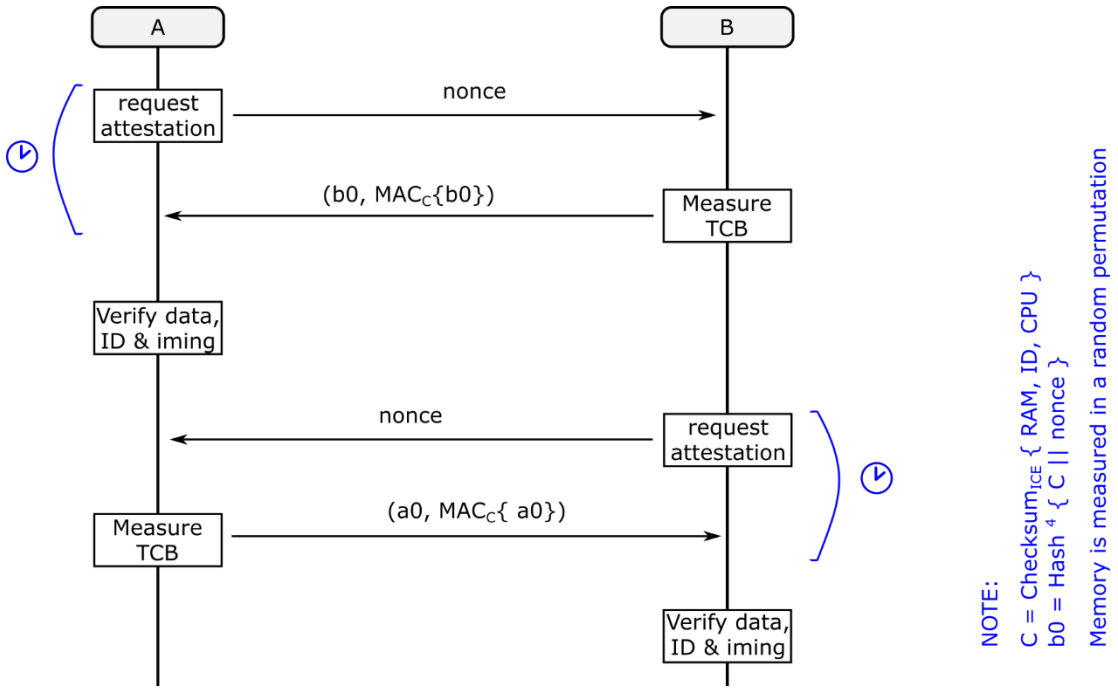

Figure 5, mutual hash commitment in SAKE.

A common issue with software-based attestation techniques is that the Prover may still be able to forge a valid response, for example by outsourcing the computation to another device. Sometimes this can be countered with additional data, for example EMMA monitors EM radiation from the Prover to detect a dishonest response. Note that this requires physical access to the device which is not always possible [16].

Another issue with these solutions is that large memory areas must be analysed which may have a significant negative effect on performance. It has however been shown that by periodically verifying subsets of the memory one can achieve attestation with a very high probability [17].

3.3.2 *Attestation on generic security hardware*

Most computing device contain some type of generic security mechanism such as a Memory Protection Unit (MPU). It would be of interest to analyse what type of attestation can be achieved using such mechanism.

SCAPI is an attestation protocol for mesh networks where a large number of devices need to be verified efficiently. A pre-shared key between each node and the Verifier is used to sign measurements while keys established between two nodes are used for communication between adjacent nodes to propagate attestation requests and responses through the mesh. All sensitive operations happen in a TEE, which in this case is some privileged code protected by an MPU [18]. slimIoT uses an MPU in a similar fashion and differs mainly using a multi-pass protocol to minimize network uncertainty in time-optimal operations [19]. When use of a standard MPU or MMU is not sufficient researchers consider security-enhanced variations, such as the execution-aware MPU in TrustLite which have additional access control mechanisms [20]. Sanctum for RISC-V extends MMU and caches to create an isolated memory region reserved for security enclaves while at the same time minimizing cache side-channel leakage [21], [22].

Attestation mechanisms generally only verify the static structure of a system, which is unfortunately left unmodified in some types of attack (e.g. ROP). C-FLAT instruments Prover

code with tracing instructions to record the actual execution at runtime. This is monitored by a measurement engine that is separated from normal software using ARM TrustZone [23].

3.3.3 *Attestation with dedicated hardware*

Similar to C-FLAT, LO-FAT is a control-flow attestation mechanism covering dynamic behaviour of applications. However, LO-FAT achieves this with dedicated hardware and therefore does not require prior instrumentation of software or a TEE [24].

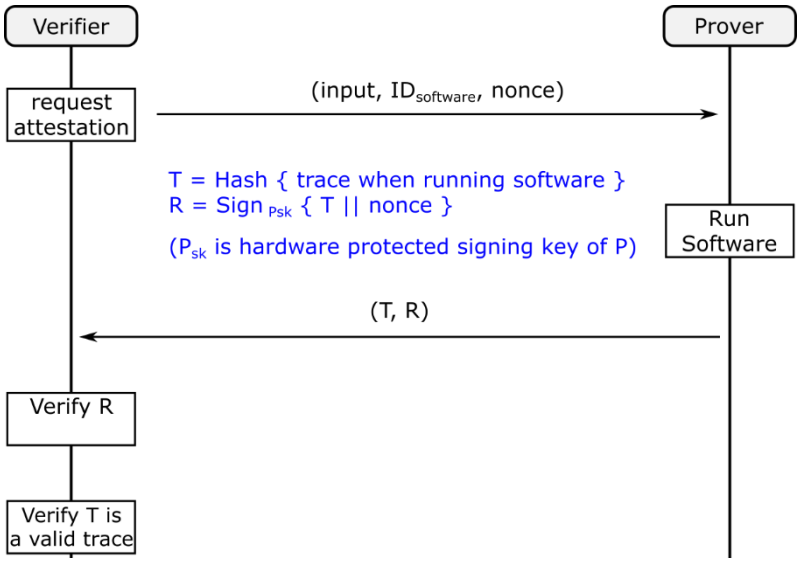

Figure 6, control-flow attestation in LO-FAT.

We have already mentioned how a TPM (or a similarly functioning HSM) can be used for attestation. The measured boot sequence illustrated in Figure 7 is an example of a *Static Root of Trust for Measurements (SRTM)* using TPM.

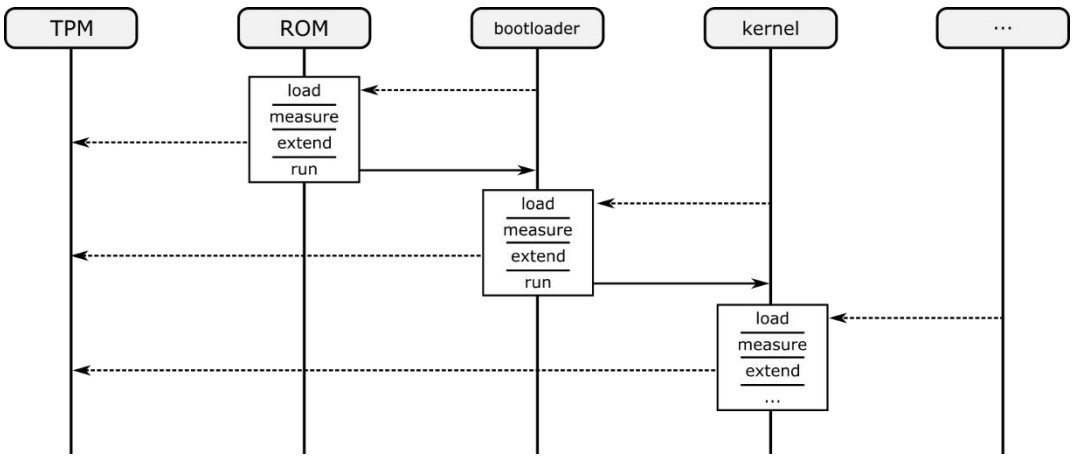

Figure 7, example of a TPM measured boot.

3.3.3.1 Intel Software Guard Extensions

Intel Software Guard Extensions (SGX) provides isolated security *enclaves*, the validity of which can be locally and remotely attested. SGX attestation is implemented in dedicated hardware plus some firmware and can be used even in presence of a malicious OS. Measurement on SGX works as follows

1. The CPU maintains a memory area not accessible by normal software
2. An (untrusted) application requests to initiate a trusted enclave
3. The CPU reserves part of this memory for the enclave, copies the enclave binary and measures it (MRENCLAVE)
4. The enclave is now active but can only interact with the application via a very narrow interface monitored by the CPU
5. Having previously measured the enclave binary, the CPU can perform local or remote attestation of its state

To perform local attestation, Intel SGX signs enclave measurement, state and configuration using a symmetric key specially derived for this purpose. This key can belong to another enclave, in which case that enclave can verify the validity of the signature:

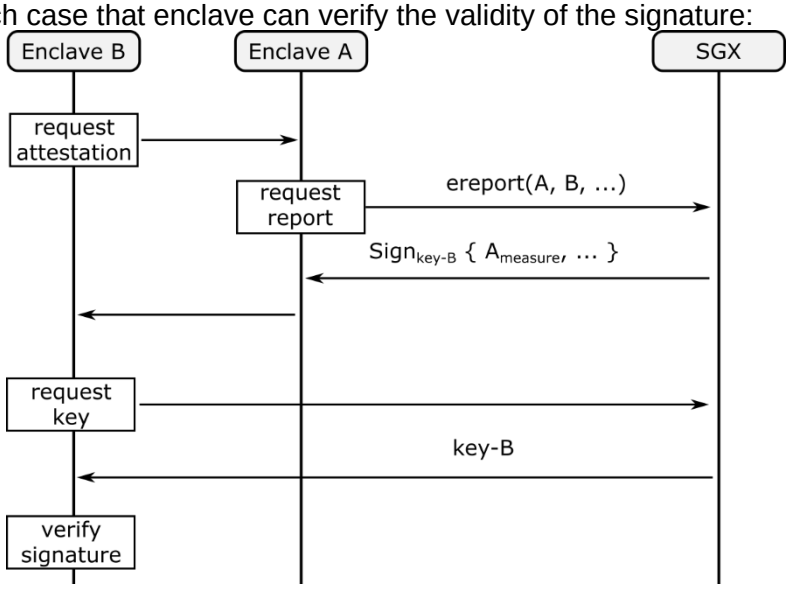

Figure 8, Intel SGX local attestation (simplified).

For remote attestation the measurement is signed by a trusted enclave (the Quoting Enclave, QE) using a group signature scheme that can be verified by a trusted third-party (i.e. Intel Attestation Service). This is illustrated in Figure 9 (details of group signature not shown).

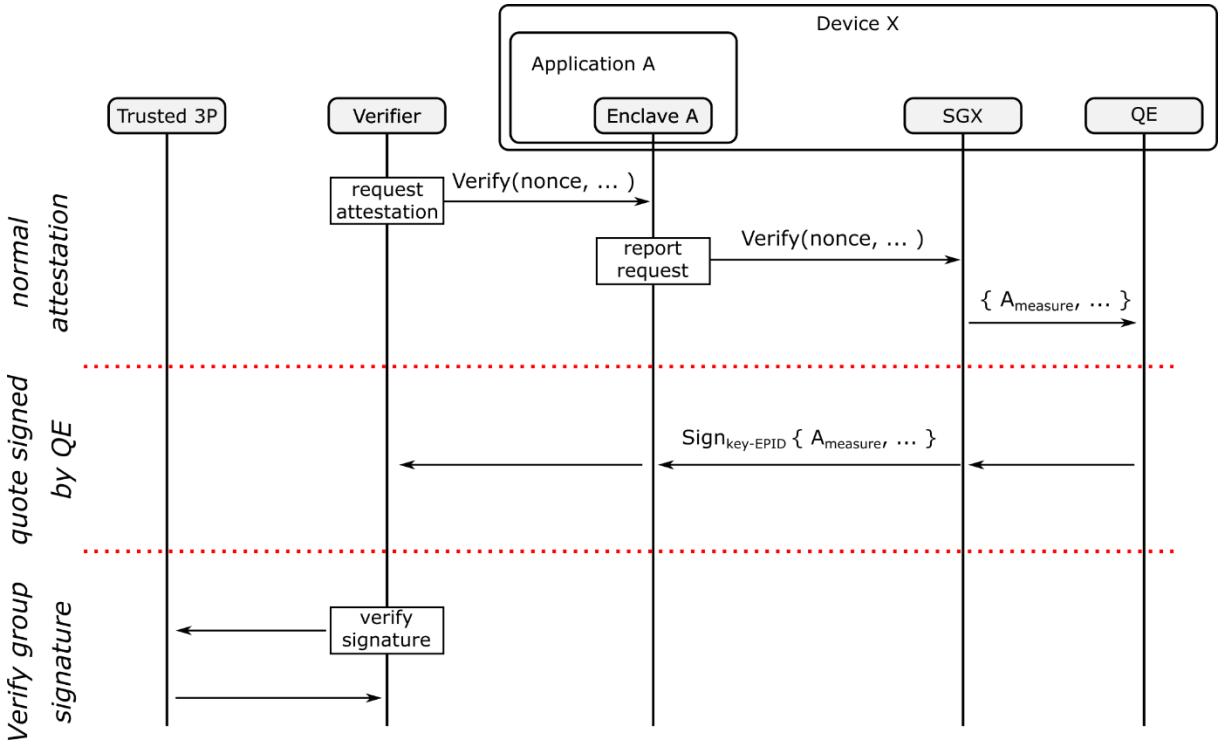

Figure 9, Intel SGX remote attestation (simplified).

3.3.3.2 AMD-SEV

AMDs Secure Encrypted Virtualization (AMD-SEV) provides mechanisms to securely deploy and execute virtual machines. Like Intel SGX, AMD uses dedicated hardware primitives to ensure trustworthiness even in presence of a malicious OS. However, unlike Intel which currently executes security firmware on the main processors (although mostly inside specially privileged enclaves), AMD uses a dedicated Security Processor (AMD-SP) for most operations.

In the past AMD processors have provided mechanisms for measured late launch using the SKINIT CPU instruction (together with a TPM). With AMD-SEV users can also deploy virtual machines using the new attestation mechanisms shown in Figure 10. With the recent ES and SNP extensions this may function even in presence of a malicious hypervisor.

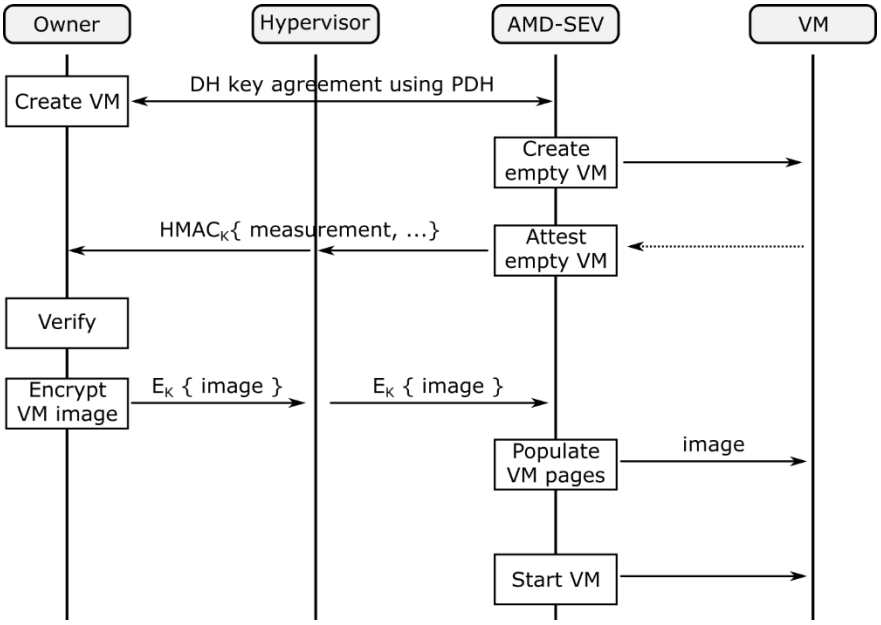

Figure 10, local attestation of VMs with AMD-SEV (simplified).

Unlike Intel SGX which utilizes group signatures, AMD-SEV uses more traditional cryptographic operations for remote attestation using the key hierarchy shown in Figure 11. The PEK is generated randomly and used for remote attestation. To verify the authenticity of this key, anyone can request the corresponding CEK_{pub} from AMD key servers.

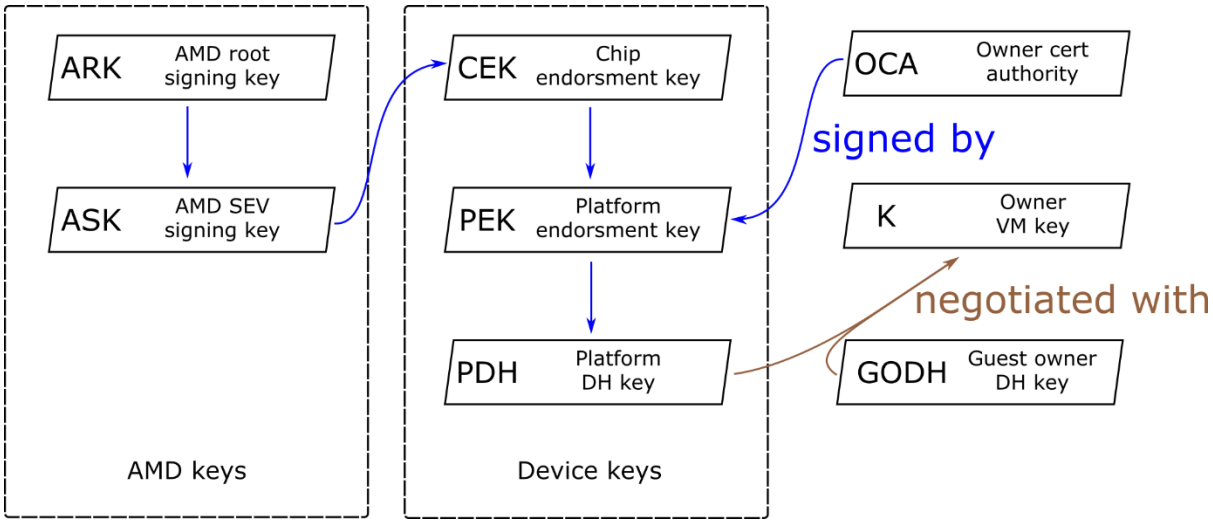

Figure 11, SEV key hierarchy (simplified).

3.4 Security issues and mitigations

It is important to understand strengths and weaknesses of each mechanisms and how/where they should be used.

Software-based attestation mechanisms are very flexible and can be implemented on most hardware. Unfortunately, they are also susceptible to more attacks, such as:

1. Pre-computation attacks: the attacker may pre-compute some or all of its response.

2. Memory-copy & memory-shadow attacks: malicious code can use free memory regions to store the original firmware.
3. Hardware modification attacks: the attacker may for example add additional memory to the device for a memory-copy attack.
4. Proxy attacks: the time optimal computations are performed on another device.

These and similar issues make use of a pure software-based attestation mechanism problematic [25]. Attestation mechanisms that use generic security hardware can provide a more secure TCB for more trustworthy operations but have their own set of issues. For example:

1. Component sharing: generic hardware security components are often shared with other functions, which may cause unintentional data leakage.
2. Size and complexity: the TCB used for attestation may be significantly larger than needed (for example an entire OS).
3. Unintended use: the threat model for local and remote attestation is often different from what these security mechanisms were originally designed for.

Attestation mechanisms that utilize dedicated hardware aim to address the above issues by using specialized hardware (and corresponding security firmware). Common issues are:

1. Lack of transparency: an independent security analysis of the mechanisms is often impossible. Users must simply trust vendors, their implementations and their services.
2. Lack of flexibility: it can be hard or even impossible to add new functionality or address discovered vulnerabilities.
3. Firmware security: the security firmware is often the main source of vulnerabilities. Sometimes critical firmware vulnerabilities cannot even be fixed with an update [26].

For example, consider the following real-world vulnerabilities (see ASCLEPIOS D4.1 for a longer list):

- CVE-2017-5691: inadequate protection of system resources allows an attacker to break the trust anchor provided by Intel SGX.
- CVE-2019-9836: insecure implementation elliptic-curve cryptography in the Secure Processor allows SEV key recovery.
- CVE-2017-16837: uncontrolled pointers in Intel's trusted boot allows malicious users to overwrite PCR registers of a TPM.

3.4.1 *Mitigations*

In D4.1 we recommended a number of measures to improve TEE security:

- Follow standard secure software development practices.
- Minimize the size of the TCB.
- Follow the principle of least privilege.
- Provide mechanisms to securely update a vulnerable TEE.

These recommendations also apply here, in addition to the following recommendations specific for attestation:

- Ensure attestation results cannot be partially or completely pre-computed (freshness).
- Ensure the complete device state can be measured if needed (completeness).
- Measurement of the device TCB should function even in presence of a malicious TCB.
- Ensure attestation can include device or group identification without jeopardizing privacy and anonymity.

Note that while data privacy is of outmost importance in ASCLEPIOS, device privacy/anonymity is not an important goal (since all participants are already known to the system infrastructure) and is included only for the sake of completeness.

4 Standardization work

The previously presented attestation solutions differ in many aspects including platform dependencies, security and use case. This could limit use of remote attestation in real-world applications. In this chapter we will look into remote attestation standardization efforts for combating this problem.

4.5 Remote Attestation Procedures Architecture (RATS)

Remote Attestation Procedures Architecture (RATS) is an upcoming IETF standard[3] that describes a common approach to remote attestation and defines a standard terminology [27].

RATS defines a number of remote attestation entities:

- **Attester** is the entity that will be appraised (evaluated/judged)
- **Verifier** is the appraising entity
- **Endorser** creates endorsement to help appraise the trustworthiness of Attester's signature
- **Relying Party** is the entity that uses the attestation result
- **Target Environment** is where the claims are gathered from. The Attester may be part of the Target Environment.
- **Owner** is the entity authorized to manage another entity

Furthermore, RATS defines a set of data items:

- **Evidence** is information about that Attester which will be verified by the Verifier
- **Attestation Result** is the attestation outcome generated by the verifier
- **Endorsement** is a secure statement that endorses Attesters signing capability
- **Policy** is a set of rules for handling and interpreting some data (for example the Replaying Party uses Appraisal Policy for Attestation Result for processing Attestation Results)

Figure 12 illustrates how these are used in RATS.

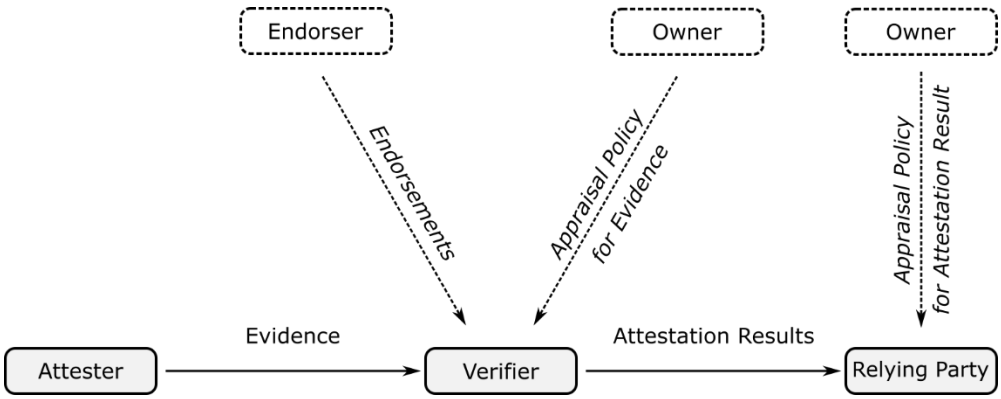

Figure 12, RATS conceptual data flow.

4.5.1 *Architectural variations*

To allow the standard to function in different scenarios and with different software and hardware targets RATS considers a number of architectural variations. For example, RATS defines two types of environments for the Attester:

1. In a **layered attester** the Attester gathers claims from multiple Target Environments
2. In a **composite device** multiple Attesters operate under a lead Attester

[I] In this document we consider draft 02 of the standard.

As an example, the TPM trusted boot is a layered attester with the layers being components of the boot sequence (firmware, bootloader, OS, …).

Communication between involved components can also happen in a number of different ways. RATS defines three in particular (two of which are shown in Figure 13):

1. In a **passport model** the Attester is responsible for providing Attestation Result
2. In a **background-check model** the Relying Party orchestrates the communication
3. In a **combined model** both are used (i.e. to handle local + remote Replying Parties)

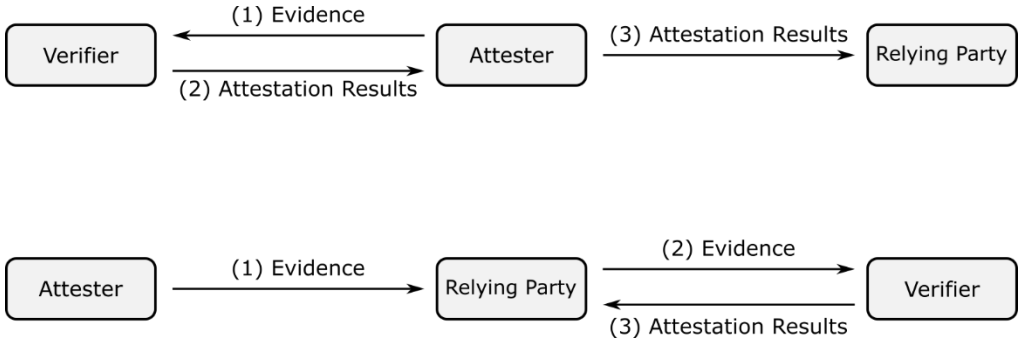

Figure 13, the passport (top) and background-check communication model (bottom).

It should be noted that these architectural variations each result in a different set of challenges for securing claims and evidences.

4.6 Requirements

RATS explores a number of requirements including security and privacy requirements that must be considered. The following is a short list of requirements we consider important for ASCLEPIOS:

1. Trust between entities, in particular trust between Replying Party and Verifier and Endorser must be established
2. Privacy must be considered, for example whether Evidence or Attestation Result contain sensitive information about the Target Environment
3. To counter replay attacks, attestation freshness must be guaranteed
4. Security of the Root of Trust is crucial and so is the integrity of the used Policies.

These are in line with the requirements and the mitigations we presented earlier.

4.7 Entity Attestation Token (EAT)

The Entity Attestation Token format (EAT) shown in Figure 14 is the IETF standard token format for attestation.

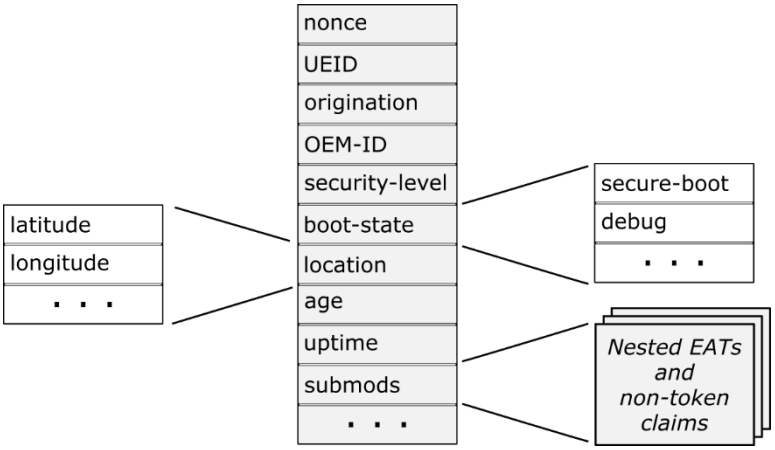

Figure 14, Entity Attestation Token (simplified).

An EAT token contains a number of mandatory and optional claims which will be signed by the Attester. EAT claims of special interest to us are:

- *nonce*, which is required to prevent replay attacks and has been generated by the Relying Party.
- *origination* is a textual description of the creator of the token.
- *security-level* is a rough characterization of the Attester security level and is selected among the following: unrestricted, restricted, secure restricted and hardware. The last three somewhat match the different attestation mechanisms we discussed earlier (software-based, generic hardware and specialized hardware).
- *boot-state* defines the boot configuration of the device and whether it has/had debug enabled.
- *submods* (submodules) contains a list of nested tokens and custom claims. For example, a claim that "name" equals "John Smith" with security-level of "unrestricted" can be included in EAT as a custom claim submod.

EAT assumes the token is signed by an attestation key material (AKM), the details of which is left to the implementer.

4.8 sTrusted Execution Environment Provisioning (TEEP)

Trusted Execution Environment Provisioning (TEEP) is an ongoing IETF standard for Trusted Execution Environments and closely related to RATS [28].

The main components of TEEP are:

- **TEEP Broker**: executes on the device but outside TEE and manages communication between that and TAM
- **TEEP Agent**: executes within the TEE and communicates with TAM, via the Broker
- **Trusted Application (TA):** the trusted application executing within the TEE
- **Trusted Application Manager (TAM):** manages the TAs

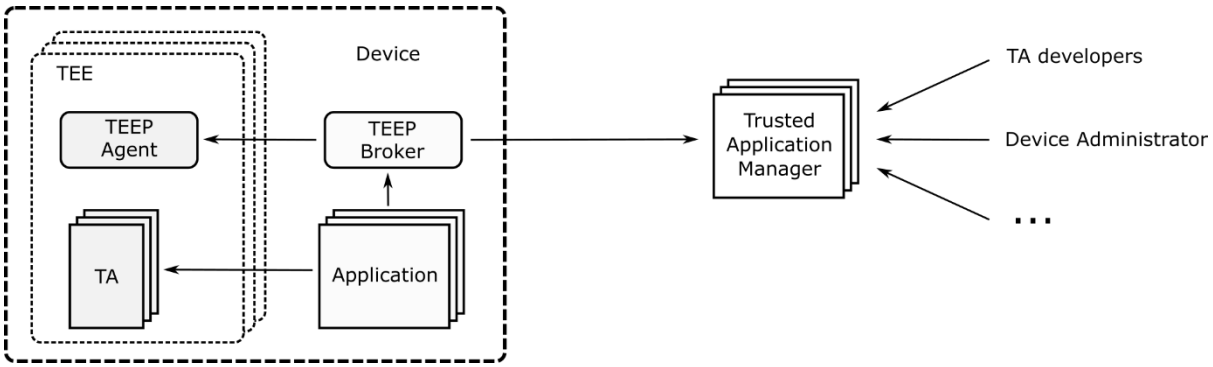

Figure 15, TEEP architecture in essence.

The standard defines a number of other important components and functions but what is of interest to this work is the support for attestation as outlined by RATS. As seen in Figure 16, the current TEEP draft considers a background-check attestation model where TAM acts as the relaying party (which may or may not be the Verifier).

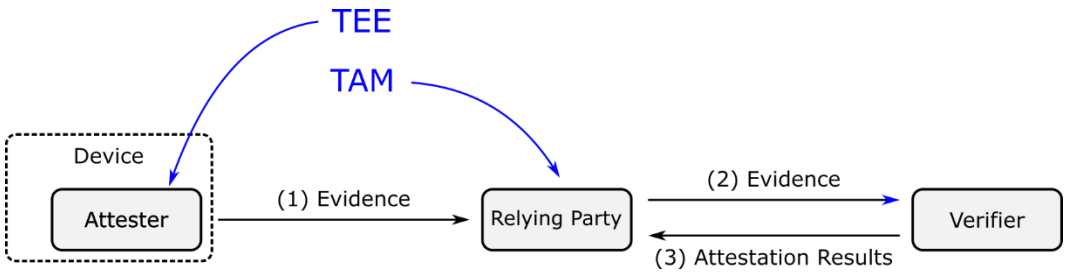

Figure 16, attestation in TEEP.

TEEP is discussed in detail in ASCLEPIOS D4.3, to which we refer readers for more information on the subject.

5 ASCLEPIOS attestation services

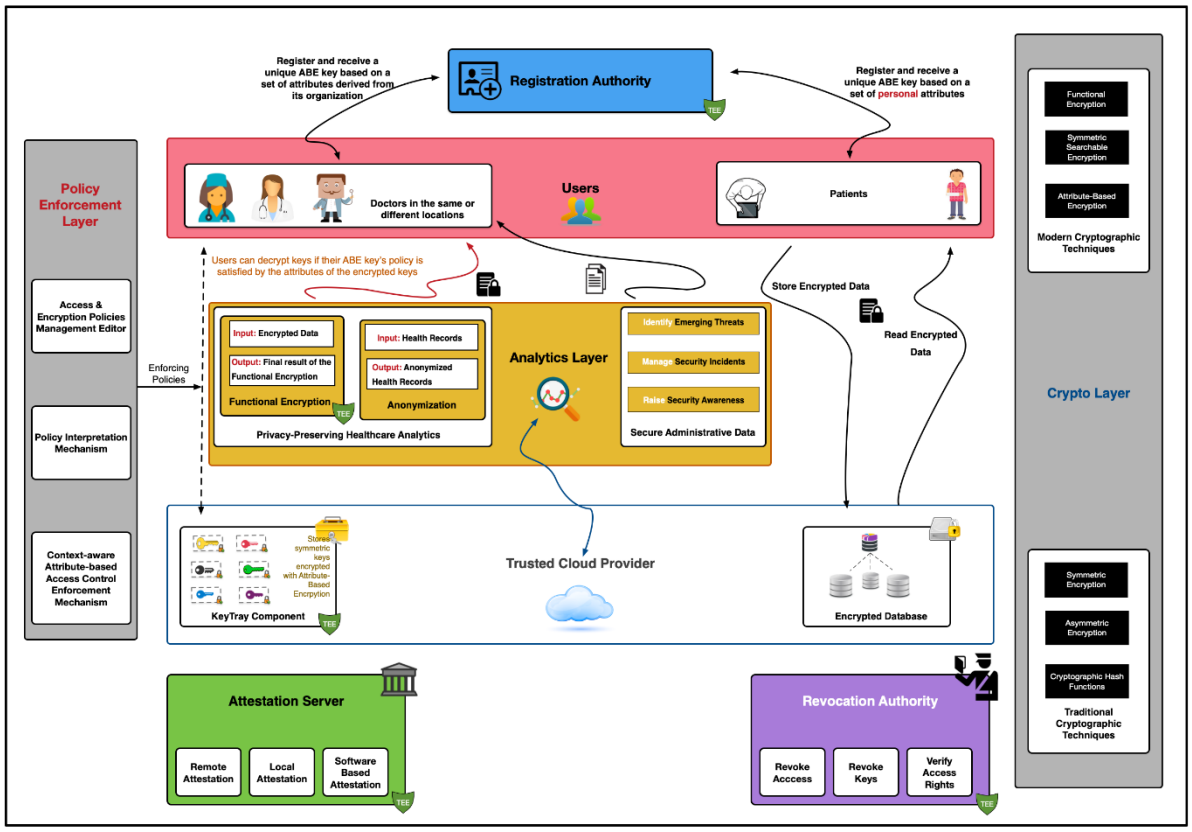

Figure 17, the ASCLEPIOS architecture.

As illustrated in Figure 17, the ASCLEPIOS architecture contains a number of core components some of which are very critical to operation and security of the entire system. For example, some devices will operate on sensitive data such as medical data or cryptographic data. To allow propagation of sensitive information between these components in a secure manner we will need to verify the authenticity and security of each party.

Note that most components in the ASCLEPIOS project contain additional security mechanisms such as cryptographic separation. However, operations such as initialization and data injection will require a secondary security mechanism.

5.9 Attestation in ASCLEPIOS

The main use of attestation in ASCLEPIOS is concentrated to a number of key areas. One area of concern for security is trust in data originating from external devices connected to ASCLEPIOS (not shown in Figure 17). These devices offer a way into the system and might be examined at leisure by attackers. It would be desirable to detect tampering and compromising (possibly by using software-based attestation) before data is accepted into the system.

Another area of concern is the Trusted Execution Environments seen in Figure 17, some of which may benefit from remote attestation while other are protected by other mechanisms. For example, the Registration Authority (RA) is accessed by users to obtain ABE keys based on some attributes. As the RA uses a form of asymmetric cryptography to generate these keys, remote attestation is not required, only proper storage of private keys and proper implementation of the cryptographic protocols.

On the other hand, consider the KeyTray which provides data decryption keys for ABE keys, in accordance to access policies defined in the Policy Enforcement Layer. The KeyTray is an essential part of the architecture that must be trusted and be secure against manipulation. If an attacker can manipulate the KeyTray to leak or incorrectly distribute keys, she may be able to use such keys to access medical data. The KeyTray is therefore considered to be essential and needs to be secured against manipulation. Preferably, attestation should be done as part of the request for symmetric keys is performed, before submitting an ABE key to the KeyTray to obtain symmetric decryption keys.

To perform attestation, the Attestation Server itself needs to be trusted. This cannot be done remotely, hence we will use more traditional methods to establish trust, such as extensive code review and real-time monitoring of server behaviour to detect tampering.

5.10 ASCLEPIOS attestation services

The ASCLEPIOS attestation server is a trusted server that acts as a CA for attestation with the key hierarchy shown in Figure 18.

In this key hierarchy two root certificates exists: The ASCLEPIOS attestation root certificate *C0* which is known by all ASCLEPIOS components and the vendor endorsement key which can be validated by the attestation server. The objective is to build a trust chain (C0-CS-CT) where each TEE can validate its workloads after initial approval from the Attestation Server.

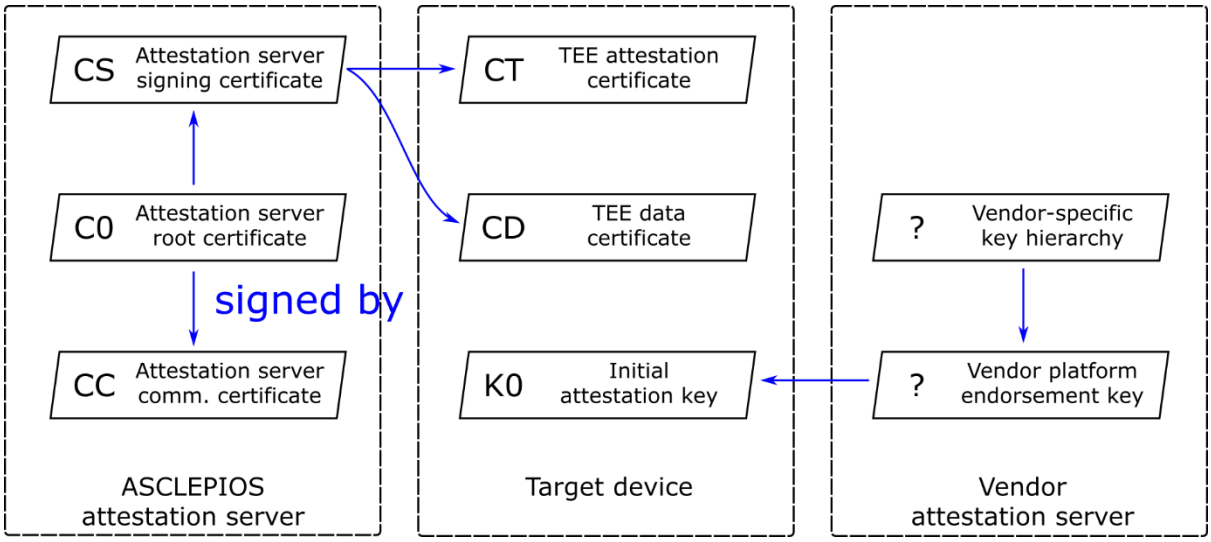

Figure 18, ASCLEPIOS attestation key hierarchy (CC and CD are reserved for future use).

The following operations are supported for attestation:

1. **GetKeys** - provides the additional certificates.
2. **Register** – registers a TEE as trustworthy after an initial remote attestation.
3. **Validate** - allows ASCLEPIOS components to submit tasks to other components and receive an attested receipt.

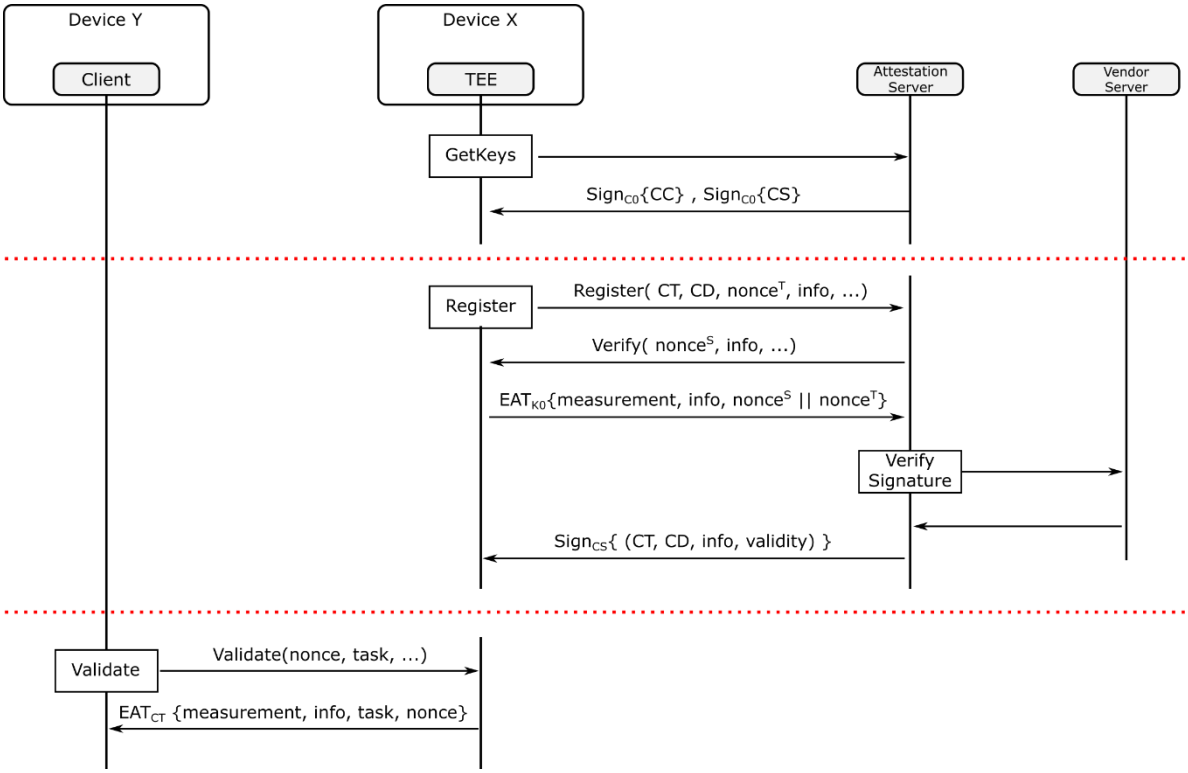

Figure 19, the ASCLEPIOS attestation operations.

The supported operations are shown in Figure 19. Note that

- The CS signature is valid only for a limited time, which in turn affects the validity of the EAT signed by CT.
- *info* contains information about the device and TEE, including identity and role (which will be verified by the recipient).
- Not included in the protocol is the logic for deciding whether device X is appropriate for a particular task.

5.10.1 *Software attestation*

The initial TEE attestation requires an evidence verified by a third party, this could for example be Intel Attestation Services or AMD Key Server.

In case of software-based attestation this authority does not exist in which case the ASCLEPIOS Attestation Server needs to judge the evidence based on other (currently unspecified) merits. Note that the strength of the evidence could affect the Attestation Server approval, for example by limiting approved roles and shortening validity time.

Note also that lack of hardware-assisted TEE also removes support for local attestation (see below).

5.10.2 *Local attestation*

We assume a TEE is present on devices that require local attestation. Local attestation will use the TEEP *QueryRequest* command and does not involve the attestation server.

We assume that - if needed - the TEE has already been remotely attested before deployment of the applications involved in the local attestation.

5.11 Security analysis

While we are unable to perform a full security analysis of the protocol at this stage, a number of key security points will be discussed in this chapter. Readers should consider what follows only as an initial approach to a future security evaluation.

We consider the following threats involving the attestation service:

- Trust: an attacker may forge the Evidence.
- Freshness: an attacker may replay old Evidence as new without detection.
- Completeness (Comprehensive Information): an attacker may modify critical software/data outside of what is included in the measurement.
- Protocol Security: an attacker may modify the attestation protocol to eventually forge or otherwise break an attestation.

We also consider a number of minor security requirements:

- Least-privilege: the attestation server should not have access to unnecessary information.
- Access Control: one should be able to verify the reach of the Target at some point during attestation.
- Limitation / Revocation: the attestation service should be able to limit the damage caused by a successful attack on a Prover or Target.

The following assets exists in this security model:

- Device attestation key material: the TEE attestation certificate (CT) used to sign Evidence.
- Server root and endorsement key material: the Attestation Server root (C0) and signing certificate (CS) are used to approve a device attestation key material (CT), the TEE role and validity.
- Vendor endorsement key material: used for initial remote attestation by an external trusted third-party.
- Random number generation: this affects the generation of nonce and any keys and certificates created on each device.

We will limit this analysis to the device and the server key material assets.

The mitigation provided to meet the security threats and security requirements are as follows:

- Trusted Execution Environment: TEE is used to maximize trust in the mechanisms used to perform attestation and to store the critical cryptographic material (Trust, Completeness)
- Challenge-Response: a nonce is used by the challenging party to ensure that the response was not recorded beforehand (Freshness)
- Composition: TEE registration requires two sets of nonce, one by each party, to ensure that an attacker cannot build a valid chain of messages from previous or forged messages (Protocol Security)
- Least-privilege: the Attestation Server does not have access to private-keys of TEEs (Least-privilege)
- Access Control: the *info* parameter contains information about the TEE and its role in the system. This parameter must be verified by the Attestation Server and the Verifier (Access Control)
- Limitation: CT signature is valid during a limited time. It can also be revoked with the help of the Revocation Server (Limitation / Revocation)

6 Summary

In this report we have investigated use of attestation in ASCLEPIOS. We have looked into different types of attestation, approaches to attestation and some implementations of those. Furthermore, we briefly explored standards related to attestation and associated technologies.

For use in ASCLEPIOS we have discussed use of remote attestation with help of dedicated security hardware (Intel SGX and AMD SEV in particular) and presented a simple protocol for using remote attestation with help of the ASCLEPIOS Attestation Service.
Finally, we have a brief look into security of the proposed solution.

The study of attestation types, approaches and implementations described in this document supports the implementation work on the Trusted Execution Environment Deployer (TEEPD) conducted throughout WP4. This work will be continued in WP5 and WP6 of the ASCLEPIOS project. In particular, TEEPD can be used to deploy ASCLEPIOS components developed for TEEs. The review of remote attestation approaches described in this document will allow to design a remote attestation implementation for ASCLEPIOS in order verify the integrity of components deployed in TEEs.

7 References

[1] J. Rushby, ”Design and Verification of Secure Systems,” *ACM Symposium on Operating System Principles,* pp. 12-21, 19831.

[2] C. Mitchel, Trusted Computing, United Kingdom: The Institution of Electrical Engineering and Technology, 2005.

[3] S. Pearson, Trusted Computing Platforms: TCPA Technology in Context, Prentice Hall Professional, 2003.

[4] S. W. Smith, Trusted Computing Platforms: Design and Applications, Springer-Verlag, 2004.

[5] A. Tomlinson, ”Introduction to the TPM,” i *Smart Cards, Tokens, Security and Applications*, Boston, MA, Springer, 2008, pp. 155-172.

[6] W. Arthur, D. Challener och K. Goldman, A Practical Guide to TPM 2.0 - Using the Trusted Platform Module in the New Age of Security, Apress Media, 2015.

[7] ”TPM 1.2 Main Specification - Part 1 Design Principles (Version 1.2, Revision 116),” Trusted Computing Group (TCG), 2011.

[8] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy och B. Sniffen, ”Principles of remote attestation,” *International Journal of Information Security,* vol. 10, pp. 63-81, 2011.

[9] E. Brickell, J. Camenisch och L. Chen, ”Direct Anonymous Attestation,” *11th ACM Conferenceon Computer and Communications Security,* pp. 132-145, 2004.

[10] E. Brickell och J. Li, ”Enhanced privacy id: a direct anonymous attestation scheme with enhanced revocation capabilities,” *ACM workshop on Privacy in electronic society,* pp. 21-30, October 2007.

[11] A. Seshadri, A. Perrig, L. V. Doorn och P. Khosla, ”SWATT: SoftWare-based ATTestation for Embedded Device,” *IEEE Symposium on Security and Privacy,* 2004.

[12] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. v. Doorn och P. Khosla, ”Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems,” *Twentieth ACM Symposium on Operating Systems Principles,* 2005.

[13] A. Seshadri, M. Luk, A. Perrig, L. v. Doorn och P. Khosla, ”SCUBA: Secure code update by attestation in sensor networks,” *ACM Workshop on Wireless Security,* 2006.

[14] R. W. Gardner, S. Garera och A. D. Rubin, ”Detecting Code Alteration by Creating a Temporary Memory Bottleneck,” *IEEE Transactions on Information Forensics and Security,* pp. 638-650, 2010.

[15] A. Seshadr, M. Luk och A. Perrig, ”SAKE: Software attestation for key establishment in sensor networks,” *Ad Hoc Networks,* vol. 9, nr 6, pp. 1059-1067, 2011.

[16] N. Sehatbakhsh, A. Nazari, H. Khan, A. Zajic och M. Prvulovic, ”EMMA: Hardware/Software Attestation Frameworkfor Embedded Systems Using Electromagnetic Signals,” *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52),* 2019.

[17] B. Chen, X. Dong, G. Bai, S. Jauhar och Y. Cheng, ”Secure and Efficient Software-based Attestation for Industrial Control Devices with ARM Processors,” *Computer Security Applications,* pp. 425-436, 2017.

[18] F. Kohnhäuser, N. Büscher, S. Gabmeyer och S. Katzenbeisser, ”Scalable Attestation Resilient to Physical Attacks for Embedded Devices in Mesh Networks,” *ArXiv 1701.08034,* 2017.

[19] M. Ammar, M. Washha, G. S. Ramachandran och B. Crispo, ”Scalable Attestation Resilient to Physical Attacks for Embedded Devices in Mesh Networks,” *IEEE Conference of Dependable and Secure Computing,* 2008.

[20] P. Koeber, S. Schulz, A. R. Sadeghi och V. Varadharajan, ”TrustLite: a security architecture for tiny embedded devices,” *European Conference on Computer Systems,* pp. 1-14, 2014.

[21] V. Costan, I. Lebedev och S. Devadas, ”Sanctum: Minimal Hardware Extensions for Strong Software Isolation,” *25th USENIX Security Symposium,* pp. 857-874, 2016.

[22] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic och D. Song, ”Keystone: An Open Framework for Architecting Trusted Execution Environments,” *Proceedings of the Fifteenth European Conference on Computer Systems,* 2020.

[23] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi och G. Tsudik, ”C-FLAT: Control-Flow Attestation for Embedded Systems Software,” *ACM SIGSAC Conference on Computer and Communications Security,* pp. 743-754, 10 2016.

[24] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan och A.-R. Sadeghi, ”LO-FAT: Low-Overhead Control Flow ATtestation in Hardware,” *54th Annual Design Automation Conference,* pp. 1-6, 2017.

[25] Y. Li, Y. Cheng, V. Gligor och A. Perrig, ”Establishing Software-Only Root of Trust on Embedded Systems: Facts and Fiction,” i *Security Protocols XXIII*, B. Christianson, P. Svenda, V. Matyás, J. Malcolm, F. Stajano och J. Anderson, Red., Springer International Publishing, 2015, pp. 50-68.

[26] R. Buhren, C. Werling och J.-P. Seifert, ”Insecure until proven updated: analyzing AMD SEV's remote attestation,” *2019 ACM SIGSAC Conference on Computer and Communications Security,* p. 1087–1099, 2019.

[27] H. Birkholz, D. Thaler, M. Richardson, N. Smith och W. Pan, ”Remote Attestation Procedures Architecture,” IETF RATS Working Group, 2020.

[28] M. Pei, H. Tschofenig, D. Thaler och D. Wheeler, ”Trusted Execution Environment Provisioning (TEEP) Architecture,” IETF TEEP Working Group, 2020.