



Project Title	ENSuring Secure and Safe CMD Design with Zero TRUST Principles		
Project Acronym	ENTRUST		
Grant Agreement No	101095634	Type of Action	Research and Innovation Action
Call / Topic	HORIZON-HLTH-2022-IND-13-01		

D4.1- Conceptual Architecture of ENTRUST Customizable TC & Attestation Models Specifications

Work Package	WP4 – Runtime Attestation Mechanisms and Continuous Certification for Post-MarketCMD surveillance		
Lead Beneficiary	QUBITECH		
Contributing Beneficiaries	SURREY, UMU, UPRC, UBI		
Due Date	31.03.24	Actual Date of Submission	24.05.24
Version	1.0		
Authors	Nada El Kassem (SURREY), Symeon Tsintzos, Alexios Askitopoulos, Niovi Sassalou (QUBI), Antonio Skarmeta, Jesus Garcia, Gilson Javier Delgado (UMU), Thanassis Giannetsos & Giannis Siachos (UBI), Ioannis Chouchoulis & Vaios Bolgouras (UPRC)		
Abstract	<p>The core vision of ENTRUST entails the secure management of the entire lifecycle of Connected Medical Devices (CMDs), starting from their manufacturing and including their deployment and operation within the infrastructure of a healthcare delivery organization. In this regard, Deliverable D4.1 puts forth the core functionalities, building blocks, and engineering stories pertaining to the components and security enablers of ENTRUST dedicated to the maintenance of the trust level of CMDs, as well as the system as a whole. Specifically, we first provide a description of the core building blocks of the Trusted Computing Base (TCB) of ENTRUST, as well as the motivation behind the design choices made for the definition of the Trusted Execution Environment (TEE) for running trusted applications in a secure and isolated manner. Next, we outline the notion of Physical Unclonable Functions (PUFs), whose unique inherent properties are leveraged by ENTRUST for safeguarding resource- constrained CMDs as a solution for obtaining identifiers and cryptographic keys to be utilized by the security enablers of ENTRUST. Considering the above, we provide high-level descriptions of the TCB security architecture of ENTRUST considering the computational capabilities of the</p>		



	<p>devices, i.e., for both high-end and low-end CMDs. Next, we outline the set of cryptographic functionalities provided by ENTRUST for securing the operational lifecycles of CMDs, including novel Attribute-Based Encryption (ABE) and Attribute-Based Signature (ABS) schemes, and we provide information on the security controls needed for establishing the trust level of a CMD during runtime. Finally, we provide functional specifications for the PUF-based functionalities employed in order to support the aforementioned security enablers. Overall, this document provides the basis for the development of the ENTRUST secure management framework, which will be refined and implemented throughout the lifecycle of the project.</p>
Keywords	<p>Customized Trusted Computing Base, Trusted Execution Environments, Physical Unclonable Functions, high-end CMDs, low-end CMDs, Trusted Extensions, Attribute-Based Encryption schemes, Attribute-Based Signature schemes</p>

Versioning and contribution history

Version	Date	Author	Notes
0.1	4.03.24	Nada El Kassem (SURREY)	Initial ToC
0.2	15.03.24	Symeon Tsintzos (QUBI), Giannis Siachos (UBI)	Provide Chapters 1,2
0.3	21.03.24	Symeon Tsintzos (QUBI), Giannis Siachos (UBI)	Provide Chapter 3
0.4	02.04.24	Thanassis Giannetsos (UBI)	Review of the ToC & additional guidelines
0.5	09.04.24	Vaios Bolgouras & Ioannis Chouchoulis (UPRC)	Provide Chapter 5
0.6	10.04.24	Nada El Kassem (SURREY)	Additional info on Chapter 5
0.7	12.04.24	Antonio Skarmeta, Jesus Garcia, Gilson Javier Delgado (UMU)	Contribution on Chapter 5
0.8	16.04.24	Giannis Siachos (UBI)	Review and additional info on Chapter 5
0.9	24.04.24	Nada El Kassem (SURREY)	Contribution on Chapters 4 and 6
0.10	25.04.24	Niovi Sassalou, Alexios Askitopoulos (QUBI)	Additional info on Chapters 4, 6, 7
0.11	20.05.24	Giannis Siachos & Thanassis Giannetsos (UBI)	Chapters 4, 6, 7 additions and Content review
0.12	17.05.24	Nada El Kassem (SURREY)	Prepare for review (figures list, tables list, cross references, address pending points in all chapters)
0.13	22.05.24	Symeon Tsintzos (QUBI)	Final technical review and refinement & Bibliography updates
1.0	24.05.24	Stavroula Isidora Giannakandropoulou (UNIS)	Final review & submission

Disclaimer

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author’s view – the European Commission is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability. This document has gone through the consortium’s internal review process and is still subject to the review of the European Commission. Updates to the content may be made at a later stage.

COPYRIGHT NOTICE

© 2023 - 2025 ENTRUST Consortium

Project co-funded by the European Commission in the Horizon Programme		
Nature of the deliverable:		*OTHER
Dissemination Level		
PU	Public, fully open	X
SEN	Sensitive, confidential to ENTRUST project and Commission Services	

* R: Document, report (excluding the periodic and final reports)
 DEM: Demonstrator, pilot, prototype, plan designs
 DEC: Websites, patents filing, press & media actions, videos, etc.
 OTHER: Software, technical diagram, etc.
 DMP: Data Management Plan

Executive Summary

Towards fulfilling the vision of ENTRUST for providing a holistic security management framework that safeguards the lifecycle of Connected Medical Devices (CMDs), this deliverable outlines the conceptual architecture of ENTRUST's customizable Trusted Computing (TC) and attestation models. ENTRUST achieves this by utilizing a novel two-fold customizable Trusted Computing Base (TCB) that covers a broad spectrum of computational capabilities, from low-end to high-end CMDs, ensuring robust trust functionalities across diverse devices.

More specifically, this deliverable presents the most recent work of WP4, focusing on the design of TC-based CMD extensions, security and trust attestation models for runtime verification, and certification and runtime auditing. The designs and schemes are documented on a high-level through the definition of Engineering Stories, which specify the technical requirements and implementation steps necessary for the project.

The deliverable provides an in-depth analysis of the Trusted Computing Base (TCB), distinguishing between low-end CMDs, which utilize Physical Unclonable Functions (PUFs), and high-end CMDs, which leverage Trusted Execution Environments (TEEs), enabling the trust establishment across the CMDs ecosystem. This analysis defines the security policies derived from extended Manufacturer Usage Descriptions (eMUDs), emphasizing confidentiality and integrity. The core functionality of Trusted Computing within ENTRUST, particularly attestation, is also explored, highlighting the innovative connection between Formal Verification and runtime attestation. This approach aids in identifying the trust boundary of CMD operations, ensuring that unverified functionalities are subject to continuous runtime attestation.

This deliverable introduces the customized ENTRUST TCB architecture, detailing its components and capabilities. It also outlines the cryptographic functionalities required for supporting the secure lifecycle of CMDs, including key management, enhanced authorization protocols, and secure communication mechanisms. The ENTRUST cryptographic toolkit is presented alongside state-of-the-art techniques, providing a comprehensive foundation for secure CMD operations. Trusted extensions and PUF-based functionalities within the ENTRUST ecosystem are also reported, presenting relevant engineering stories and practical insights. The document also describes crucial implementation aspects of these elements into the ENTRUST framework, ensuring seamless and secure CMD management.

Overall, this deliverable serves as a critical foundation for the subsequent developments in WP4, providing a clear and detailed roadmap for implementing the specified security and trust mechanisms. The outlined architecture and technical designs will guide the development and refinement of ENTRUST's secure lifecycle management framework, ensuring robust and reliable security for CMDs.

Contents

1	Introduction	3
1.1	Scope and Purpose	4
1.2	Deliverable Structure	4
1.3	Development Roadmap	5
1.3.1	Milestones.....	6
1.3.2	Design Phase	6
1.3.3	Predeployment Phase	6
1.3.4	Secure Enrollment Phase	7
1.3.5	Runtime Phase	8
2	Introduction to Trusted Computing	10
2.1	Introduction to Trusted Computing.....	10
2.1.1	What is Trusted Computing Base (TCB)?.....	10
2.1.2	Minimizing the Trusted Computing Base (TCB)	10
2.1.3	What is a Root of Trust (RoT)?	11
2.1.4	What is a Trusted Execution Environment (TEE)?	11
2.2	State of the Art on Trusted Computing	12
2.2.1	State-of-the-Art on HW-based Roots-of-Trust.....	12
2.2.2	State-of-the-Art on Trusted Execution Environments (TEEs).....	13
2.2.3	Run-time Environments for TEEs	15
2.3	Building a Trusted Computing Base Leveraging PUFs.....	16
2.3.1	Introduction to PUFs.....	16
2.3.2	PUFs classification	17
2.3.3	ENTRUST PUF instances	18
2.4	Motivation in ENTRUST	21
3	The Customized ENTRUST TCB	22
3.1	ENTRUST CTCB Functionalities.....	22
3.2	Trusted Computing Base Building Blocks	23
3.2.1	ENTRUST TEE Design Choice.....	23
3.2.2	SRAM-PUFs in the context of ENTRUST CTCB.....	24

3.3	High-Level Description over the ENTRUST's TCB Security Architecture.....	33
3.3.1	High-End CMDs TCB Overview.....	33
3.3.2	Low-End CMDs TCB Overview.....	35
3.3.3	Engineering Stories Supported by ENTRUST's Dynamically Adjustable to the type of the device TCB	35
4	Cryptographic Functionalities for Supporting the Secure Lifecycle of CMDs	37
4.1	ENTRUST Cryptographic Toolkit	38
4.2	State of the Art.....	39
4.2.1	Enhanced authorization protocol for medical device on-boarding	39
4.2.2	ENTRUT Key Management.....	40
4.2.3	Crypto primitives and protocols for continuous secure and authenticated communication	42
4.3	Engineering stories	44
5	Trusted extensions	61
5.1	State of the Art.....	61
5.1.1	Tracing techniques.....	61
5.2	Engineering stories	62
6	ENTRUST hybrid PUF-based Element Functionalities	84
6.1	Engineering stories	84
7	Conclusions	88
	Bibliography	90

List of Figures

1.1	Timeline for WP4-related activities	5
2.1	Compromised OS in an untrusted environment may leak sensitive data while it is in use. . .	12
2.2	Arm TrustZone architecture (from [https://dl.acm.org/doi/10.1145/3308755.3308761]) . . .	14
2.3	Schematic representation of a physical unclonable function (PUF). The token (sometimes also referred to as PUF tag), is a device with internal physical disorder. The internal disorder of the token is imprinted into its response to a physical challenge. The raw response is processed classically in order to yield a nearly perfect and robust random key. (from [51]) .	17
2.4	The circuit diagram for a Static Random Access Memory (SRAM) cell	19
2.5	Schematic representation displaying the operating principle of a speckle imaging-based optical PUF when a typical disordered medium is utilized (a) and when phase masks generated by an Spatial Light Modulator(SLM) (b), are utilized as PUF "tokens".	20
3.1	Secure Storage System Architecture of OP-TEE	24
3.2	PUF-based system identification threshold for Intra and Inter (a) Non-overlapping, (b) Overlapping	26
3.3	Generation and reproduction procedure of Fuzzy Extractor.....	27
3.4	The experimental setup considered here for the characterization of the proposed SRAM-based PUF implementation.....	31
3.5	The SRAM modules' crucial characteristics in terms of FHD	32
3.6	ENTRUST security stack for high-end CMDs	34
3.7	ENTRUST security stack for low-end CMDs.....	35
3.8	Engineering stories supported by ENTRUST's customized TCB mapping	36
4.1	Key Hierarchy	41
4.2	Attribute Based Access Control (ABAC)	43
4.3	Flow for Story-II: As a SP I want to have the necessary guarantees of a device's trust properties of interest when authenticated and registered into an operational domain	46
4.6	Secure Enrolment	52
4.7	Flow for Story-VI: As a high-end CMD, I want to be able to securely retrieve the Protection Profile of the CMD from the ledger.....	54
4.8	Flow for Story-VII: As a low-end CMD, I want to be able to securely communicate with the	



	Domain Manager and collect the required types of evidence directly	56
4.4	Flow for <i>Story-III: As a security administrator, I want to be able to securely enroll the device to the domain manager in an autonomous manner</i>	59
	Flow for <i>Story-IV: As a user I want to have the feature of data sovereignty resolving of full control of who can access my medical records</i>	60
5.1	Flow for <i>Story-XII: As a SP I want to be able to have assurances on the correct configuration and behavioral integrity of a device</i>	65
5.2	Flow for <i>Story-XIII: As a Trust Controls module, I want to be able to audit the device status (during runtime in a verifiable manner)</i>	69
5.3	Flow for <i>Story-XIV: As a SP I want to be able to get access to runtime device system measurements in case of an indication of risk</i>	70
5.4	Flow for <i>Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device</i>	73
5.5	HS SPEC	81
5.6	HS SPEC	83
6.1	Authentication of low-end CDM utilizing high-end device leveraging high-entropy cryptographic primitives generated from a SRAM-based PUF instance	8

List of Tables

4.1	Flow for Story-II: As a SP I want to have the necessary guarantees of a device's trust properties of interest when authenticated and registered into an operational domain	47
4.2	Flow for Story-III: As a security administrator, I want to be able to securely enroll the device to the domain manager in an autonomous manner	49
4.3	Flow for Story-IV: As a user I want to have the feature of data sovereignty resolving of full control of who can access my medical records	51
4.4	Domain Enrolment	53
4.5	Flow for Story-VI: As a high-end CMD, I want to be able to securely retrieve the Protection Profile of the CMD from the ledger.....	54
4.6	Flow for Story-VII: As a low-end CMD, I want to be able to securely communicate with the Domain Manager and collect the required types of evidence directly	57
4.7	Revocation	58
5.1	Flow for Story-XII: As a SP I want to be able to have assurances on the correct configura-tional and behavioral integrity of a device.....	66
5.2	Flow for Story-XIII: As a Trust Controls module, I want to be able to audit the device status (during runtime in a verifiable manner)	68
5.3	Flow for Story-XIV: As a SP I want to be able to get access to runtime device system measurements in case of an indication of risk.....	71
5.4	Flow for Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.....	72
5.5	Flow for Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.....	78
5.6	Flow for Story-XVII: As a Gateway Tracer I want to protect the integrity and authenticity of the gathered data	80
5.7	Flow for ??: As a Bare Metal Tracer I want to protect the integrity and authenticity of the gathered data	82
6.1	Swarm Attestation	87

List of Abbreviations

Term	Description
AAA	Authentication, Authorization, and Accounting
ABAC	Attribute-based Access Control
ABE	Attribute-based Encryption
ABS	Attribute-based Signatures
ACL	Access Control List
AK	Attestation Key
ATL	Actual Level of Trust
CA	Certification Authority
CFA	Control-Flow Attestation
CIV	Code Integrity Validation
CMD	Connected Medical Device
CVE	Common Vulnerabilities and Exposures
DID	Domain Identifier
DLT	Distributed Ledger Technology
DT	Digital Twin
FV	Formal Verification
FW	Firmware
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoC	Indicator of Compromise
IoT	Internet of Things
MUD	Manufacturer Usage Descriptions
NIST	National Institute of Standards and Technology
PUF	Physical Unclonable Function
RA	Risk Assessment
RoT	Root-of-Trust
RTL	Required Level of Trustworthiness
SA	Static Analysis
SL	Subjective Logic
SoS	System of Systems
SPLC	Secure Product Life Cycle
SSI	Self-Sovereign Identity
SW	Software
TAR	Trust Assessment Request
TARA	Threat Agent Risk Assessment
TC	Trusted Component

TDE	Trust Decision Engine
TEE	Trusted Execution Environment
TI	Threat Intelligence
TLEE	Trustworthiness Level Expression Engine
TM	Threat Modelling
TMM	Trust Model Manager
TSM	Trust Sources Manager
UID	Unique Initial Identification
VC	Verifiable Credential
VP	Verifiable Presentation

Chapter 1

Introduction

In the context of the Secure Lifecycle Management of Connected Medical Devices (CMDs), one of the core pillars of ENTRUST's vision, we aim to be able to associate the trust level of CMDs for every point in time based on the principles zero trust. We identified four phases that enable this goal, namely, the Design Phase, the Pre-deployment Phase, the Deployment Phase, and the Runtime Phase. During the Design Phase, the CMD is located in the manufacturer premises and its core security functionalities are designed. The pre-Deployment phase consists of the CMD secure boot-up and the assessment/identification of Risks for the device in the context of its operational domain. During the Deployment Phase, the CMD is onboarded to the service graph chain and its enablers for runtime attestation are constructed. Finally, during Runtime Phase, the CMD continuously attests its configurational state.

One main goals in ENTRUST, is to maintain an up-to-date trust quantification for the entire service graph chain comprised by the CMDs, elevated from the trust state of each device. To that end, the CMDs should be equipped with an interoperable security stack that allows the monitoring of trustworthy evidence, originating from the security controls (intrusion detection systems, network misbehavior mechanisms, at- testation) that inspect the CMDs. For the operation of runtime trust assessment, the core enabler that can make this ambitious vision possible, revolves around the insights from the academic field of Trusted Computing.

NIST defines the Trusted Computing Base (TCB) as *“The totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination of which is responsible for enforcing a security policy”*. In ENTRUST, the policies are originating from the eMUDs (see D3.1), clearly outlining security objectives in terms of confidentiality and integrity, i.e., what the medical device is expected to protect against, such as unauthorized access or data leakage. Regarding the set of software and primitives that constitute the TCB, in ENTRUST, we differentiate between two types of devices: low-end and high-end CMDs. Low-end CMDs have a lightweight TCB, constituting of a Physical Unclonable Function (PUF) while, on the other hand, high-end CMDs have a richer TCB, constituting of a Trusted Execution Environment (TEE).

The core functionality of Trusted Computing is attestation. The main challenge here, which still remains open in both remote and runtime attestation, is the identification of the properties that need to be attested. One of the core offerings of ENTRUST towards addressing this challenge is the correlation of the Formal Verification with runtime attestation. The term Formal Verification refers to the systematic Validation and Verification actions of ENTRUST for providing guarantees on the correctness of SW and HW co-designs

developed, supported by well-established methods and tools. Formal Verification aids in the identification of the Trust Boundary of the operation of a CMD, thus, the functionalities that can't be formally verified, comprise the behavioral profile of the device that has to be attested during runtime.

1.1 Scope and Purpose

As already mentioned, the vision of ENTRUST is to provide a holistic security management framework for the secure lifecycle of CMDs. ENTRUST accomplishes this by utilizing a novel two-fold customizable TCB that covers a large spectrum of the computational capabilities of various devices. Deliverable 4.1, "Conceptual Architecture of ENTRUST Customizable TC and Attestation Models Specifications" documents the most recent work of the following tasks, under the umbrella of WP4:

- Task 4.1: Design of TC-based CMD Extensions
- Task 4.2: Security and Trust Attestation Models for Runtime Verification
- Task 4.3: Certification and Runtime Auditing

The objective of the Task 4.1 is to design the novel trusted computing architecture for the secure configuration, operation and verifiable computing of CMD nodes. In this task, the two types of TCBs that will be utilized in ENTRUST were identified, covering high-end and low-end CMDs. Task 4.2 defines the type of attestation mechanisms to be leveraged for calculating the security claims that need to be verified for assessing the level of trustworthiness of a medical device. The output of this task so far, includes the identification of the various attestation mechanisms that will support a CMD during its lifecycle. Finally, Task 4.3 defines the process that needs to be followed for the certification, based on the security claims outputted by Task 4.2. Here, the flow of actions for the creation and update of Verifiable Credentials and Presentations and their binding with the eMUDS was designed.

The main purpose of deliverable D4.1 is to present the engineering blueprints (called Engineering Stories (ES)) that describe the set of cryptographic functionalities, trust extensions and PUF-based TCB specifications for securing the lifecycle of CMDs. Also, a state of the art in Trusted Computing and a first description of ENTRUST's TCB is provided. D4.1 is the first deliverable of the WP4 deliverable series and describes the aforementioned functionalities in a high-level, with the help of Engineering Stories. In the upcoming deliverables (D4.2, D4.3), a concrete and updated description of those functionalities will be presented.

1.2 Deliverable Structure

The deliverable is organized as follows:

- **Chapter 1** introduces the scope and purpose of the document, providing an overview of the development roadmap.
- **Chapter 2** introduces the concept of trusted computing and provides a State Of The Art in the field. This chapter further elaborates on building a TCB leveraging Physical Unclonable Functions (PUFs), detailing PUF classification and specific ENTRUST PUF instances, and the motivation behind the ENTRUST project.

- **Chapter 3** describes the customized ENTRUST TCB, detailing its functionalities and the building blocks of the trusted computing base. It discusses the ENTRUST TEE design choices and the role of SRAM-PUFs within the ENTRUST TCB.
- **Chapter 4** focuses on cryptographic functionalities for supporting the secure lifecycle of constrained medical devices (CMDs). It presents the ENTRUST cryptographic toolkit and the state-of-the-art in this domain, including enhanced authorization protocols for medical device onboarding, key management, and crypto primitives and protocols for secure communication. This chapter also includes engineering stories to provide practical insights.

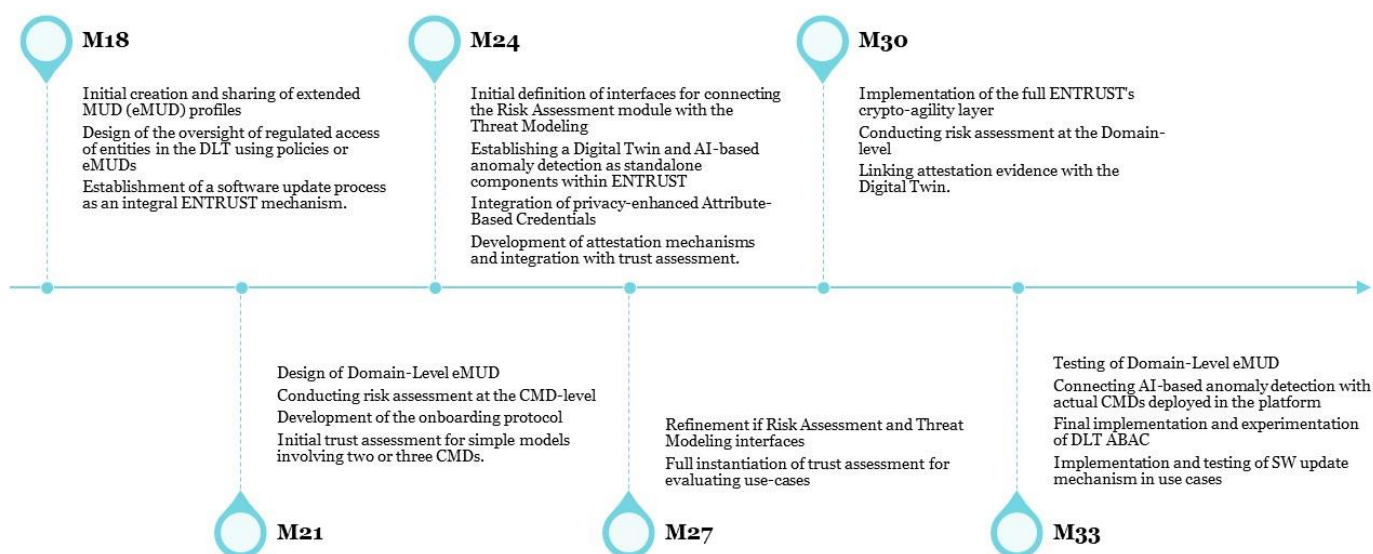


Figure 1.1: Timeline for WP4-related activities

- **Chapter 5** examines trusted extensions, reviewing the state-of-the-art tracing techniques and presenting relevant engineering stories.
- **Chapter 6** explores ENTRUST hybrid PUF-based element functionalities, providing an introduction and related engineering stories.
- **Chapter 7** concludes the deliverable, summarizing the key findings and contributions of the project.

1.3 Development Roadmap

This section outlines the research and development plan of the ENTRUST consortium, focusing on the technical objectives of WP4. The primary goals include:

- Design and development of the ENTRUST customizable TC-based middleware offering novel trust extensions.
- Development of new efficient trust attestation mechanisms.
- Definition of continuous runtime certification by extending Manufacturer Usage Descriptions (MUDs) to include runtime security claims on device assurance level.

To achieve these goals, the development process is divided into two phases, each with specific milestones. We envision a two-round experimentation period to iteratively refine and validate our components. The complete vision for the Development is depicted in Figure 1.1.

1.3.1 Milestones

The project is structured around key milestones to ensure steady progress and timely delivery of components. The milestones are:

- **M18:** Initial design of ENTRUST's components.
- **M21:** First version of ENTRUST's framework.
- **M24:** First round of experimentation.
- **M27:** Refined design of ENTRUST's components.
- **M30:** Second version of ENTRUST's framework.
- **M33:** Second round of experimentation.

The first three milestones constitute the 1st release of the ENTRUST platform, while the others form the 2nd. Those milestones are aligned with the two-round implementation methodology of ENTRUST, as highlighted in the Description of the Action.

1.3.2 Design Phase

During this phase, we will focus on creating and extending MUD profiles, which are crucial for ensuring secure and trusted communication among Connected Mobile Devices (CMDs).

Extended MUD Profiles

The major part of the work in the design phase is to create extended MUD profiles for CMDs and share them with all relevant stakeholders through the Distributed Ledger Technology (DLT). This will involve two distinct tasks:

- **1st release of the ENTRUST platform:** Initial creation and sharing of extended MUD (eMUD)

profiles.

- **2nd release of the ENTRUST platform:** Extending the eMUD and leveraging it as a building block for the Domain-level MUD. This includes incorporating risks and policies during CMD onboarding, as detailed in D4.2.

1.3.3 Predeployment Phase

Following the design phase, the predeployment phase prepares the system for deployment by conducting thorough risk assessments and establishing necessary interfaces.

Risk Assessment

Risk assessment is critical to ensure the security and reliability of the CMDs. This will be conducted at multiple levels:

- **1st release of the ENTRUST platform:** Conducting risk assessment at the CMD-level to identify potential vulnerabilities and mitigate risks early on.
- **2nd release of the ENTRUST platform:** Conducting risk assessment at the Domain-level to ensure comprehensive security across the entire ecosystem.

Interfaces for Risk Assessment and Threat Modeling

To enhance the effectiveness of the risk assessment framework, we will define interfaces that connect the Risk Assessment module with the Threat Modeling module:

- **1st release of the ENTRUST platform:** Initial definition of interfaces for connecting the Risk Assessment module with the Threat Modeling module. This will facilitate better coordination between risk assessment and threat modeling processes.
- **2nd release of the ENTRUST platform:** Refinement and finalization of these interfaces based on experimentation outcomes, ensuring robust and integrated risk management.

Establishment of Digital Twins

Digital Twins will play a crucial role in monitoring and analyzing the behavior of CMDs. The establishment of Digital Twins will be carried out in two stages:

- **1st release of the ENTRUST platform:** Establishing a Digital Twin and AI-based anomaly detection as standalone components within ENTRUST. This will enable initial monitoring and anomaly detection capabilities.
- **2nd release of the ENTRUST platform:** Connecting AI-based anomaly detection with actual CMDs deployed in the platform. This integration will enhance real-time monitoring and anomaly

detection.

1.3.4 Secure Enrollment Phase

In the secure enrollment phase, we will focus on developing a robust onboarding protocol and integrating advanced cryptographic mechanisms to ensure secure and seamless CMD integration into the ENTRUST ecosystem.

Onboarding Protocol

The onboarding protocol is essential for authenticating CMDs and establishing secure communications. This will involve:

- **1st release of the ENTRUST platform:** Development of the onboarding protocol, including authentication and construction of Verifiable Credentials (VCs) and Verifiable Presentations (VPs). This will lay the groundwork for secure CMD integration.
- **1st release of the ENTRUST platform:** Integration of privacy-enhanced Attribute-Based Credentials (p-ABC) for issuing conformity certificates. This will enhance privacy and security during CMD onboarding.
- **2nd release of the ENTRUST platform:** Implementation of the full ENTRUST's crypto-agility layer featuring signcryption, runtime construction of VPs, and authentication between low-end and high-end CMDs. This will ensure flexible and robust security mechanisms.

1.3.5 Runtime Phase

The runtime phase focuses on ensuring secure and trustworthy operations of CMDs within the ENTRUST ecosystem. This includes implementing access control mechanisms and conducting ongoing trust assessments.

Two-Layer Access Control for DLT authentication

Effective access control is crucial for regulating the interaction of entities within the DLT. In the initial version, the DLT will manage attestation data and policies without access control, providing a foundation for future enhancements. This basic setup will be crucial for achieving a secure and trusted operational environment in subsequent versions. This will be addressed through:

- **1st release of the ENTRUST platform** Design of the oversight of regulated access of entities in the DLT using policies or eMUDs in D4.2.
- **2nd release of the ENTRUST platform** Final implementation and experimentation.

Trust Assessment

Trust assessment is vital for maintaining the integrity and reliability of the system. This will be developed in stages:

- **1st release of the ENTRUST platform:** Initial trust assessment for simple models involving two or three CMDs. This will establish basic trust metrics.
- **2nd release of the ENTRUST platform:** Full instantiation of trust assessment for evaluating use-cases. This will provide comprehensive trust evaluation for various scenarios.

Attestation Mechanisms

Developing robust attestation mechanisms is key to ensuring CMD integrity and trustworthiness. This will be achieved through:

- **1st release of the ENTRUST platform:** Development of attestation mechanisms and integration with trust assessment. This will establish the initial attestation framework.
- **2nd release of the ENTRUST platform:** Linking attestation evidence with the Digital Twin. This will enhance the monitoring and verification of CMD integrity.

SW update process as an ENTRUST mechanism

The software update process is essential to maintain the security and functionality of CMDs over time. This will involve:

- **1st release of the ENTRUST platform:** Establishment of a software update process as an integral ENTRUST mechanism. This will ensure CMDs can securely receive updates.
- **2nd release of the ENTRUST platform:** Implementation of this process in ENTRUST's use-cases.

By following this structured development roadmap, the ENTRUST consortium aims to achieve its technical objectives and deliver a robust, secure, and trustworthy system for securely managing CMD's lifecycle.

Chapter 2

Introduction to Trusted Computing

2.1 Introduction to Trusted Computing

2.1.1 What is Trusted Computing Base (TCB)?

End users of a secure system often rely on critical services of a system. Even more so, in “*Systems-of-Systems*”, the availability and safety of an integral component may depend on the correctness and availability of certain services. To correctly provide such services, the architecture must ensure that certain critical components always behave as expected by the users or else fail safely. Because misbehavior of services that guarantee given requirements of the user often cannot be detected by the users themselves, this is called the *Trusted Computing Base* (TCB) [71].

In general, the TCB may include communication, storage, and computation. Software developers may use encryption to protect *data in transit* and *data at rest*. For example, remote users can connect to the application server via secure network channels using SSL/TLS and VPN. All data traveling through these network channels are encrypted while in transit, such that a potential attacker cannot learn any secrets. In another example, software typically encrypts data before writing it to the hard disk, so that even if attackers gain access to private files, they cannot discern the stored secrets.

Formally, as introduced earlier, we define a **”TCB for a given requirement of a service or a function” as the set of hardware, firmware, services, and/or software components that are required to function correctly in order to guarantee that the requirement is met.**

Following this definition, different requirements on a service may require different TCBs. For instance, while the TCB for the confidentiality of a real-time mapping service may include the components used for end-to-end encryption, the TCB for the availability of the same service may include a hot-spare system that allows fail-over at run-time without interrupting the service.

Note that it is best practice to consider the TCB at design time. TCBs can be designed to be static or extensible. A static TCB is designed to protect a fixed set of requirements and cannot be extended. E.g. a Trusted Platform Module (TPM) chip provides fixed functions to allow integrity protection by means of attestation.

2.1.2 Minimizing the Trusted Computing Base (TCB)

The TCB can become very large, but it is a well-known empirical fact that the larger a component is, the more bugs it can have [58]. As a consequence, there is a higher risk of defects. Furthermore, by definition, a single bug in the TCB can compromise the given requirement that it protects. This constitutes a significant risk for critical services.

To mitigate these risks, designers aim to *minimize the TCB*. In particular for (security) critical services, designers usually try to minimize the size of the TCB. For instance, a small hardware security module for storing keys usually has a lower risk of compromise compared to a large cloud service providing the same function.

2.1.3 What is a Root of Trust (RoT)?

The (hardware) Root of Trust (RoT) is the minimal set of security guarantees – usually provided as built-in hardware capabilities – that is sufficient to protect TCBs. The design goal is to ensure the protection of a (usually software-based) TCB under the assumption that a well-defined set of security functions are provided by the (usually hardware) RoT.

Again, the RoT can be large, such as a whole microcontroller in a TPM, or can be smaller, like an encryption engine with loaded keys. One advantage of a hardware Root-of-Trust (RoT) is that it usually cannot be modified by its owner. As a consequence, it can provide a mechanism to build trust with a remote user; i.e., the remote user must gain trust that their application running in an untrusted environment is executed as expected and that the application secrets are protected. To this end, hardware-based TEEs provide *remote attestation* capabilities.

2.1.4 What is a Trusted Execution Environment (TEE)?

A Trusted Execution Environment (TEE) provides a secure and isolated environment for performing critical activities, such as computations, that must be executed with a high degree of assurance. The concept which the TEE is based on, is the distinction between the “*trusted*” and the “*untrusted*” world of the host where the TEE is instantiated. By separating the two worlds, a TEE manages to create a safe environment that protects against unauthorized access as well as disclosure or tampering of confidential information. A Trusted Execution Environment can protect the confidentiality and integrity of enclosed, loaded code and data. In other words, TEEs provide a **confined, isolated domain** in which the application runs, and this domain appears completely opaque to other software running on the same server. TEEs are one type of technology that can serve as a Root-of-Trust for supporting the secure execution of safety-critical binaries.

One goal of a TEE is to remove the untrusted and large operating system from the TCB: Once the size of the TCB has been minimized and separates the software TCB from the underlying hardware RoT, the TCB often still includes the operating system and most of the hardware (e.g. memory and storage). To further enhance the security of the TCB, designers then (a) minimize the required trust in the hardware, and (b) further minimize the software that is required to be trusted by removing the operating system from the TCB. *But why is it important to protect an application from a compromised operating system?*

Typically, software needs to manage various types of secrets, such as encryption keys, authentication credentials, financial information, and so on. These days, software developers offload their software to

run on public clouds or in other untrusted environments. Second, a minimal TCB with only the trusted application and underlying hardware is enough to protect the application from a compromised operating system. However, when data is used in the actual computation happening on the untrusted server (*data in use*), it is processed unencrypted within the CPU. In other words, data is completely in the clear as soon as it moves into the memory of the server. If the attacker controls the server – either by installing malware or simply by virtue of having physical access to the server – the attacker can steal application secrets. Figure 2.1 illustrates this.

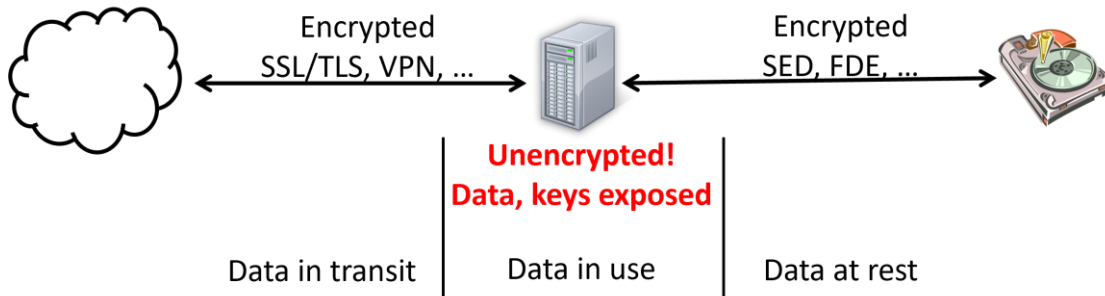


Figure 2.1: Compromised OS in an untrusted environment may leak sensitive data while it is in use.

2.2 State of the Art on Trusted Computing

2.2.1 State-of-the-Art on HW-based Roots-of-Trust

To protect a software TCB, well-defined hardware security guarantees are essential. This is where the term **trusted computing** has been introduced, providing technologies and proposals for resolving computer security problems through hardware enhancements. The Trusted Computing Group (TCG) is an industrial standards organization that aims to create TPM specifications. Its main objective is to develop, define, and promote open, vendor-neutral, global industry specifications and standards, supportive of a hardware-based RoT, for inter-operable trusted computing platforms. To that end, there is a need for trusted hardware. Examples of such hardware solutions include Trusted Platform Module (TPM), Trusted Execution Environments (TEEs), Device Identifier Composition Engine (DICE) and Physically Unclonable Function (PUF). In the context of ENTRUST, TEEs and PUFs are considered, thus special focus will be given in the following subsections.

Trusted Platform Module (TPM): The TPM standard defines a hardware chip. When a TPM is embedded in its host platform, it serves as the Root of Trust for measurement and report. The TPM is also used as a cryptographic engine, which supports general cryptographic functionalities, such as digital signatures, encryption and message authentication. It also supports secure and flexible key management solutions, for example, it can implement light-weight multiple-layers key hierarchy with a very small amount of internal memory. With the key hierarchy architecture, the TPM can securely store cryptographic key material, which helps build security solutions in IoT. The nature of hardware-based cryptography ensures that the information stored in the TPM is better protected than software-preserved data. The TPM serves as a trust anchor for a host platform it is embedded in. It creates proof attestations about the state of the host system, e.g., certifying the boot sequence the host is running on.

Hardware Security Modules (HSM): A Hardware Security Module (HSM) is a specialized, tamper-resistant device designed to securely store cryptographic keys and perform cryptographic operations [2]. HSMs provide a dedicated hardware environment for key management and cryptographic functions, offering a high level of security against various forms of attacks, including physical tampering and software vulnerabilities. HSMs play a critical role in securing sensitive data and communications in industries such as finance, healthcare, government, and cloud computing. They are used to protect against threats such as data breaches, fraud, and unauthorized access by providing a trusted execution environment for cryptographic operations. One of the key features of HSMs is their compliance with industry standards.

and regulatory requirements. For example, many HSMs are certified under the Federal Information Processing Standards (FIPS) 140-2, a standard issued by the National Institute of Standards and Technology (NIST) that specifies security requirements for cryptographic modules [10]. Compliance with standards such as FIPS 140-2 ensures that HSMs meet rigorous security standards and can be trusted to protect sensitive information. HSMs also offer advanced key management capabilities, including key generation, distribution, rotation, and backup. These features enable organizations to securely manage cryptographic keys throughout their lifecycle, ensuring that keys are properly protected and can be securely accessed when needed. Furthermore, HSMs provide high-performance cryptographic operations, making them suitable for use in environments that require fast and efficient encryption and decryption. By offloading cryptographic operations to dedicated hardware, HSMs help improve the overall security and performance of applications and systems [4].

Device Identifier Composition Engine (DICE): DICE stands for Device Identifier Composition Engine. The engine is embedded in hardware in the first immutable boot loader that loads the first component off-chip. DICE is suited for embedded devices and is a lightweight solution compared to other HW-based (TPM) or SW-based (Intel SGX) solutions that are widely used today. At the same time, the DICE is embedded within the same chip, thus, unlike TPMs allows for a threat model that includes the chip boundary only. The Device Identifier Composition Engine is based on the first level of mutable code and Unique Device Secret that can be embedded by the owner of the device (or the manufacturer). The assumption in the DICE is that the first mutable code is a boot level that does not change and thus, the engine will always deduce the keys based on 1) Unique Device Secret UDS or 2) hash of first boot level. DICE enables a solution that does not rely on strict hardware but allows a family of hardware and software techniques that rooted back in hardware. The DICE using the hardware and software techniques provide Root of Trust for reporting (attestations), Root of Trust for storage (data encryption), and other deterministic seed for cryptographic functions. DICE security standard is created by the Trusted Computing Group (TCG) and supported by Microsoft's open-source project RIoT core [28], which acts as the first boot level security extension that does not change during runtime. DICE Architecture is a simple and new security approach suit for Internet of Things (IoT) and embedded devices that does not increase silicon requirements and provide the ability to owners to embed their own secret keys and own the entire attestation solution and identity of the device. The DICE architecture, with its hardware Root of Trust for measurement, breaks the software stack up into layers and creates unique secrets and a measure of integrity for each layer which means that any point in time, it can create conformity certificates of integrity for each software layer/process.

2.2.2 State-of-the-Art on Trusted Execution Environments (TEEs)

A wide range of TEEs have been proposed recently [52] - from simple HSMs like AUTOSAR SHE¹ to complete shielded and tamper-proof subsystems (e.g. the IBM4765 cryptographic co-processor). Here, only TEE functionalities provided by commercial off-the-shelf CPUs since in the context of ENTRUST an integrated approach where non-security and security parts co-exist on the same hardware platform is considered. Thus, in this subsection we will describe in short the most prominent TEE flavours of this type, namely ARM TrustZone and Intel SGX.

In general, TEEs consist of several components:

- **Secure Bootstrapping** to ensure the system starts at a secure initial state.
- **Secure Input/Output** to provide an end-to-end path for secure communication with network, storage, and other peripheral devices.
- **Isolated Execution** to protect data in use.
- **Remote Attestation** to prove the trustworthiness of TEE to a remote party.
- **Secure provisioning** to obtain secrets from a remote party.

¹https://www.autosar.org/fileadmin/standards/R21-11/FO/AUTOSAR_TR_SecureHardwareExtensions.pdf

ARM TrustZone: TrustZone [8] is Arm’s System-on-Chip and CPU system-wide security solution, available on Arm application processors and present in the new generation Arm microcontrollers. TrustZone follows a System-on-Chip (SoC) and CPU system-wide approach to security. This technology is centered around the concept of protection domains named secure world and normal world, but there is a remarkable difference regarding the processor used: on Cortex-A processors, the privileged software referred by the name of secure monitor implements mechanisms for secure context switching between worlds (by means of a special instruction called “secure monitor call” (smc)). This secure monitor is in essence similar to Keystone’s, at least in terms of privilege modes. However, on Arm Cortex-M processors, there is no secure monitor software and the bridge between both worlds is handled by a set of mechanisms implemented into the core logic. Regardless the existence of a security monitor (or lack thereof), the basic operation relies on a strong hardware-enforced separation between worlds. This makes it possible to constrain the OS operations to the normal (untrusted) world, while securing critical applications in the secure world; thus providing security guarantees without trusting complex OS like e.g. Android.

For I/O devices and interrupts, TrustZone also splits them into two worlds with asymmetrical privileges. In addition to this, in a TrustZone architecture, the TEE kernel leverages secure boot to enforce its own TEE to memory and checks its signature integrity. This happens at boot time when the processor enters.

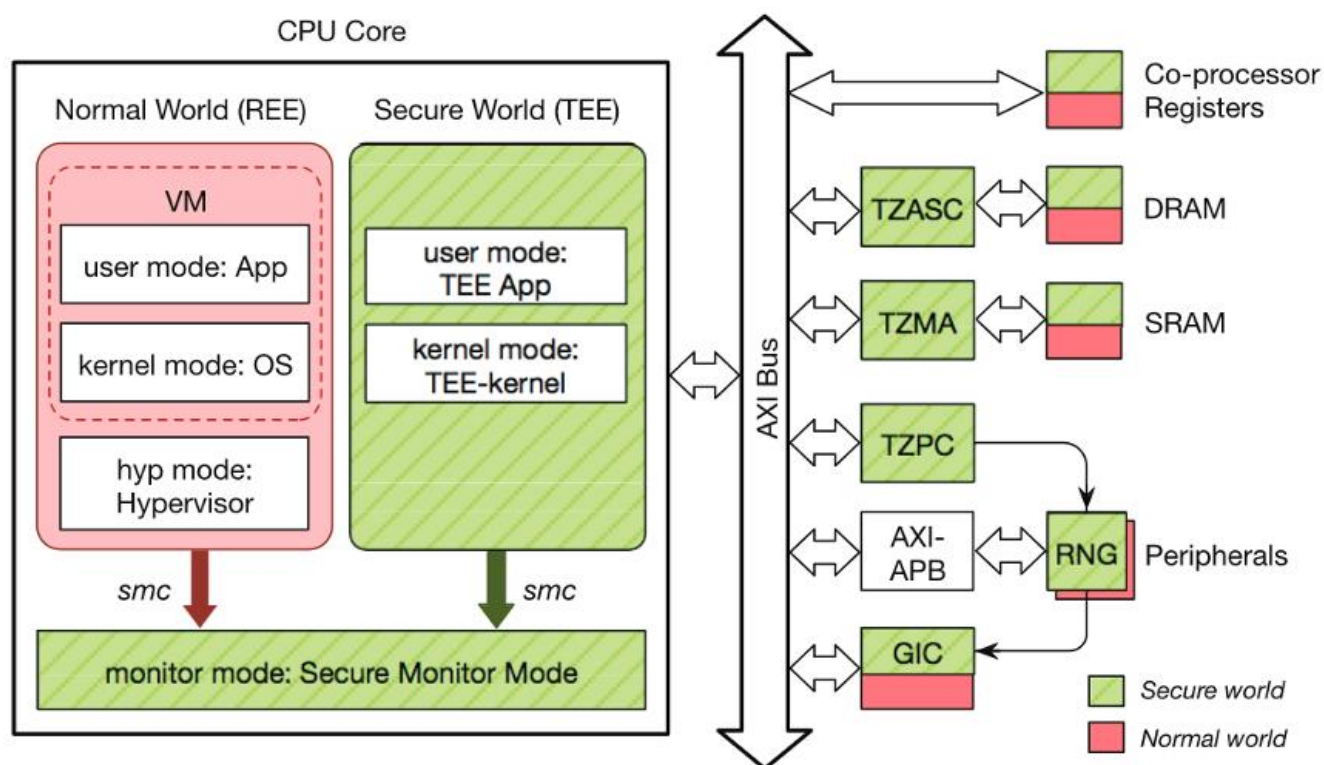


Figure 2.2: Arm TrustZone architecture (from [https://dl.acm.org/doi/10.1145/3308755.3308761])

Arm's own policies for IP protection [72] historically made difficult for developers to understand and evaluate the security strengths – and risks – of TrustZone, but this changed around a decade ago. A shift in attitude by major hardware manufacturers, namely Xilinx, meant an important factor in helping the rising interest in TrustZone. Xilinx publicly disclosed TrustZone technical details and releases documentation and development boards [3]. This has facilitated things for researchers studying the security properties of TrustZone-based systems, uncovering on the other hand important vulnerabilities.

For example [43], the Boomerang attack leverages bugs of privileged TEE applications to compromise the REE (normal world) kernel. Another consideration of TEE security, is the update process. An attacker may insert malicious logic to TEE images to be updated, or (downgrade attack) it may try to revert the system to a previous firmware version that is known to contain certain vulnerabilities for further exploitation. Other forms of attack that threaten TrustZone (and other TEE implementations) are side channel and physical attacks. Many SoCs provide customized protection mechanisms on memory and peripherals for this reason. For instance, the key storage and functions inside some Samsung smartphones are kept inside the Keymaster trustlet (secure world), which uses AES-256 in GCM mode (making cache attack against this target much harder), but it has been proved possible [39] to perform a successful cache attack against this AES implementation using widely available hardware.

Intel SGX: Intel SGX [24] is a feature of Intel CPUs. Unlike TrustZone, it does not partition the complete CPU into trusted/untrusted but allows each user-space application to declare so-called "enclaves" that protect a subset of an application. The CPU then provides dedicated protections of integrity and confidentiality of this enclave.

Intel SGX enables user-level code to allocate exclusive memory locations, known as enclaves, that are

specifically designed to be protected from programs operating at higher privilege levels. Applications can communicate with the enclave by invoking a trusted function, which can be run within a secure enclave. The enclave only allows authorized functions to execute, and any attempts to access enclave data are denied by the processor. Hardware-level encryption protects against software-based attacks, ensuring sensitive data remains undisclosed even if a hacker gains control over the operating system and BIOS. Enclaves may autonomously produce and store their own exclusive signing/attestation keys, thereby excluding any external party from accessing them. Data may exclusively be signed by utilizing keys that are linked to certain instruction sets, operating within each enclave.

A simple and illustrative example is a Browser that can partition-out its built-in password manager into a dedicated enclave. The CPU then ensures that the password manager is confidentiality and integrity protected. I.e. the memory and storage of the password manager is encrypted. The OS and applications cannot read the passwords, except the access provided by the API that is implemented inside the enclave. Furthermore, if an attacker then tries to launch a modified password manager to export the passwords, the CPU will refuse to launch it or will not allow this modified password manager to gain access to the encrypted passwords.

Note that one challenge when using SGX is that the Operating System (OS) still has strong influence over the enclave and thus writing code that remains secure even if attacked by the OS is challenging. Attack frameworks like SGXstep [70] allow the adversary outside the enclave to exercise fine-grained control which may result in leakage of secrets.

2.2.3 Run-time Environments for TEEs

Trusted Execution Environments (TEEs) inherently consist of hardware features designed to facilitate secure execution. However, due to the intricacies and complexities of directly interfacing with CPU hardware features, application developers often encounter challenges in harnessing the full potential of TEEs. To address this issue, a diverse array of software frameworks has been proposed, aiming to streamline the development, deployment, and debugging processes of TEE applications. These frameworks abstract away much of the complexity involved in TEE development, making it easier for developers to build secure and trusted applications while also providing tools for provisioning and debugging to ensure the integrity and confidentiality of TEE-based solutions. An indicative list of such environments is presented below:

1. **Open Enclave SDK:** An open-source framework developed by Microsoft for building TEE applications. It supports various TEE backends, including Intel SGX and Arm TrustZone. Open Enclave SDK provides APIs and runtime libraries to simplify development and debugging of enclave-based applications
2. **OP-TEE (Open Portable TEE):** OP-TEE is an open-source TEE project that provides a comprehensive TEE solution. It offers a runtime environment and development tools to simplify TEE application development and provisioning.
3. **Intel SGX SDK:** Provided by Intel, the SGX SDK offers libraries and tools to aid in the development and debugging of SGX enclave applications. It simplifies the process of creating secure enclaves and integrating them into applications.
4. **SCONE (Secure Container Environment):** SCONE is an open-source framework for securing containerized applications with Intel SGX. It provides tools and libraries for developers to easily integrate

SGX-based security into containerized applications.

5. Occlum: Occlum is an open-source TEE runtime environment that supports both Intel SGX and Arm TrustZone. It provides a POSIX-compliant execution environment within a secure enclave, simplifying application development for TEEs

6. Graphene-SGX: Graphene-SGX is a lightweight library OS for Intel SGX that allows unmodified applications to run securely within SGX enclaves. It simplifies the migration of existing applications to the SGX platform.

7. Panoply: Panoply is a library OS-based TEE framework that provides a runtime environment for running applications securely within Intel SGX enclaves. It aims to simplify the development and deployment of enclave-based applications.

2.3 Building a Trusted Computing Base Leveraging PUFs

2.3.1 Introduction to PUFs

Physically unclonable functions (PUFs), as shown in Fig.2.3, were introduced in 2002 [34] as a security primitive based on hardware. PUF leverages inherent manufacturing variations within a device to create a distinct hardware fingerprint (sometimes referred as token or PUF tag), offering the significant advantage of unclonability. Essentially, a PUF is the physical analogue of a one-way mathematical function, based on an unclonable, non-reproducible and complex physical mechanism. Combined with their deterministic operation, PUFs are appropriate for cryptographic key generation on demand, eliminating the need for key storage (no key-at-rest property). This characteristic distinguishes PUFs from other hardware-based security approaches, since someone, even with physical access to the device, cannot replicate its intrinsic properties. Consequently, PUFs are uniquely tied to their respective devices, serving as a security primitive for enabling device-centric identification and authentication. Moreover, PUFs present a cost-effective alternative for generating cryptographic keys on demand directly from the device, deviating from traditional methods where secret keys are generated and distributed by servers and stored in the memories of IoT devices [19]. PUFs utilize the inherent variations inherent in the manufacturing process of the device (e.g. a SRAM, a multiplex) or a component (sandblasted silica glass) to derive a unique fingerprint specific to that device or component.

During the measurement process, one or more specific parameters of the device, such as threshold voltage or critical dimensions, are assessed when an external stimulus is applied. This initial measurement, known as the "original response," is obtained for a particular input stimulus or a designated memory address referred to as a "challenge." Both the original response and the corresponding challenge are stored in a trusted node or component for future reference. Subsequently, when the same parameter is measured again under identical conditions, it yields a response. These pairs of challenges and responses constitute what is termed a **Challenge-Response Pair (CRP)** and are typically compared with each other to authenticate the identity of the device. The nature of the challenge depends on the PUF under consideration, while the response to a given challenge depends strongly on the internal disorder of the device. The raw noisy response is classically processed by means of a fuzzy extractor to yield a nearly perfect robust binary key. The fuzzy extractor typically involves two separate processes: reconciliation and hashing. The former aims at a reconciled key that is not affected by environmental

variations and ageing of the token, whereas the latter compresses the reconciled key further, so that the final key is nearly uniformly distributed. According to [18], a hardware module to be qualified as PUF, should exhibit the following characteristics:

1. **Robustness:** A response to the same challenge should be able to be reproduced over time and over a various range of conditions.
2. **Unpredictability:** A response to a challenge on a PUF device should be unrelated to a response to another challenge from the same device or the same challenge from a different device.
3. **Unclonability:** CRPs mapping of a device should be unique and cannot be duplicated even from the device manufacturer per se.
4. **Physically Unbreakable:** Any physical attempts to maliciously modify the device will result in malfunction or permanent damage.

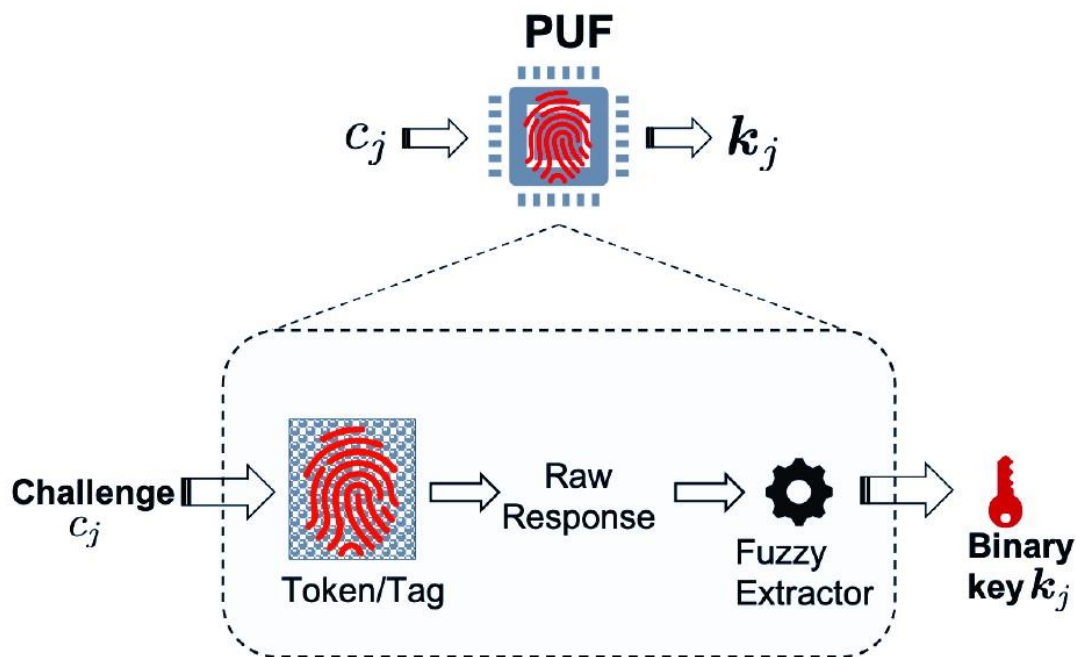


Figure 2.3: Schematic representation of a physical unclonable function (PUF). The token (sometimes also referred to as PUF tag), is a device with internal physical disorder. The internal disorder of the token is imprinted into its response to a physical challenge. The raw response is processed classically in order to yield a nearly perfect and robust random key. (from [51])

2.3.2 PUFs classification

Subject to the number of possible CRPs a PUF has, they can be broadly classified into: “*strong PUFs*” and “*weak PUFs*”. The latter leverage the manufacturing variability and allow digitization of some “fingerprint” of the hardware device. The number of responses in a weak PUF is a function of the number of components in the device (e.g. numbers of cells in case of memory based PUFs) used for generation of CRPs. This fact results in a small number of CRPs with stable responses which are usually robust to

environmental conditions. Due to high stability and reproducibility of weak PUF responses, they are generally used for **secret key generation**. On the contrary, “strong PUFs” have a large number of CRPs in a device. Ideally, if the number of unique CRPs is high, even though an attacker gets temporary accesses to the system, he/she will not be able to apply all the responses (brute force attack) and get access to the system. Hence, strong PUFs are generally used for **authentication**.

Besides the number of CRPs, PUFs can also be categorized based on their physical design which dictates the source of randomness. There are two major categories, *explicit* and *implicit* PUFs [62]. In explicit PUFs, the unique randomness is obtained by externally applying additional steps, e.g. coating PUFs where specific materials or size of particles are used to introduce randomness. However, in such a case the location and the distribution of this randomness cannot be easily controlled. Some other examples of this type of PUFs are optical PUF, RF-DNA PUF, optical fiber PUF and acoustic PUF.

In implicit PUFs, the origin of the randomness is natural, through variations in the manufacturing process. There are two subcategories in implicit PUFs, delay based and memory based PUFs. An example of delay based PUF is arbiter PUF. The main principle of arbiter PUF is by presenting a race condition on two different routes on a chip where the winner will be decided by an arbiter circuit. As in memory based PUFs, some examples of this design are SRAM PUF, butterfly PUF and latch PUF. SRAM PUF utilized the random physical mismatch in the cell introduced by manufacturing variability which controls the power-up behavior (can be zero, one, or no preference). Butterfly PUF uses the effect of cross-coupling between two transparent data latches. Using the functionalities of the latches, an unsteady condition can be initiated after which the circuit resolves back to one of the two stable states. In latch PUF, the concept is based on using two NOR gates which are cross-coupled. These gates will lead to a stable condition depending on the internal discrepancy between the electronic components [62].

2.3.3 ENTRUST PUF instances

Medical Devices (MDs), functioning as integral components within the Internet of Things (IoT) ecosystem, exhibit distinctive characteristics defined by their constraints on processing power, memory capacity, and often reliance on battery power. Moreover, MDs are typically subject to stringent regulatory standards within the medical and healthcare industries, ensuring compliance with safety regulations. Consequently, any form of interference at the device level is strictly prohibited, given the specific certifications they must adhere to.

Addressing security concerns at both the software and hardware levels presents considerable challenges within this context. Traditional approaches to hardware-level security are constrained, with current options primarily limited to hardware security modules that execute a predefined set of cryptographic operations, such as key management, key exchange, and encryption, directly at the processor level. However, these solutions are predominantly applicable to modern MDs, leaving existing devices without suitable protection mechanisms.

An alternative approach to bolstering hardware-level security in MDs lies in the adoption of PUFs. PUFs offer the potential to enhance hardware security by injecting randomness into the key generation process during secure device onboarding procedures. They can also facilitate identity management and cryptographic operations, particularly when designing advanced attestation schemes [63] aimed at fortifying the overall security posture of MDs. By leveraging the inherent physical variations within each

device, PUFs serve as a promising avenue for strengthening the security of medical devices, thereby mitigating vulnerabilities and safeguarding sensitive healthcare data.

The two most commonly recognized types of PUFs, categorized by their fabrication processes, are optical (photonic) and electronic. The former are considered strong [53], whereas the latter are classified as weak PUFs [64]. Within the ENTRUST framework, both types will undergo comprehensive testing, tailored to meet the specific challenges posed by medical devices, as well as the technical and use case requirements, documented in D2.1. Emphasis will be placed on silicon PUFs, considering their size, cost, and implementation flexibility. Concurrently, optical PUFs (o-PUFs) will be explored, leveraging their outstanding security attributes stemming from the probabilistic procedures involved in deriving keys from high-entropy optical mediums (such as photonic chaotic cavities or disordered random optical scattering media) commonly utilized. In the case of silicon PUFs, a **static random access memory-based** approach was chosen, while for non-silicon PUFs, a **speckle imaging**-based optical PUF was considered.

• Static Random Access Memory (SRAM) PUF

The SRAM PUF uniquely characterises a system through the variation of otherwise symmetric transistor branches within static random access memory (SRAM) elements, as a result of the variation in the manufacture process. A SRAM memory element, as featured in Fig. 2.3, consists of a collection of inverters and access transistors such that there are two stable states at a certain input power (a bistable flip-flop circuit). When power is applied, the cell can be written into either state. The system is kept stable in this state and can later be read as memory. When the circuit is unpowered both states are low, but when power is initially applied to the cell without additional bias the system stabilises at one of the two stable states. Here, the challenge is the address (position) of the SRAM element and the response is this “power up state” of the element.

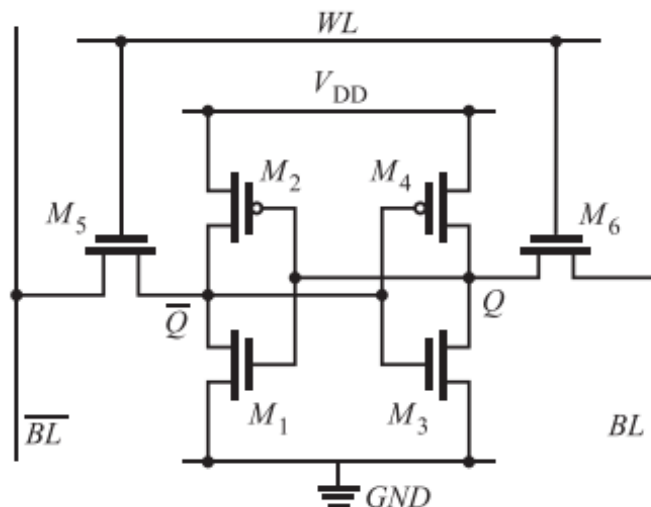


Figure 2.4: The circuit diagram for a Static Random Access Memory (SRAM) cell

Many notable variations of the SRAM PUF concept exist, fulfilling a variety of special purposes and solving a variety of emergent issues. These include flip-flop, latch, butterfly, and buskeeper PUFs. These PUFs rely on the same bistable principle that SRAM PUFs use, where for certain input voltage two potential arrangements of currents and voltages across the circuit are stable. They exist as more advanced storage

elements that mitigate issues with the original SRAM setup or adapt the SRAM for different uses. For example, a Butterfly PUF is a variation on a SRAM PUF designed for programming with FPGAs. A cell of a Butterfly PUF is much alike that of a SRAM PUF; however, in most common FPGAs, SRAM cells are hard reset to zero (and thus all randomness is lost) directly after powerup. The cells of this PUF are constructed from cross-coupling two transparent data latch cells. These converge in a manner comparable to SRAM cells after power-up but without being explicitly SRAM elements. In the case of a SRAM cell, a power-up is required to engage in response generation, which similarly is not necessary with the butterfly PUF cell.

- **Speckle Imaging-based Optical (photonic) PUF**

The original optical (photonic) PUF (o(p)-PUF) [53], of eponymous nomenclature, relies on the interaction of a coherent light source (challenge) with a disordered microstructure. The propagation of light through this medium is inherently complex, and thus unpredictable. In the original form, a laser is shone through an “optical token,” a plate of microscopic refractive particles mixed into an epoxy plate as shown in Fig. 2.5(a).

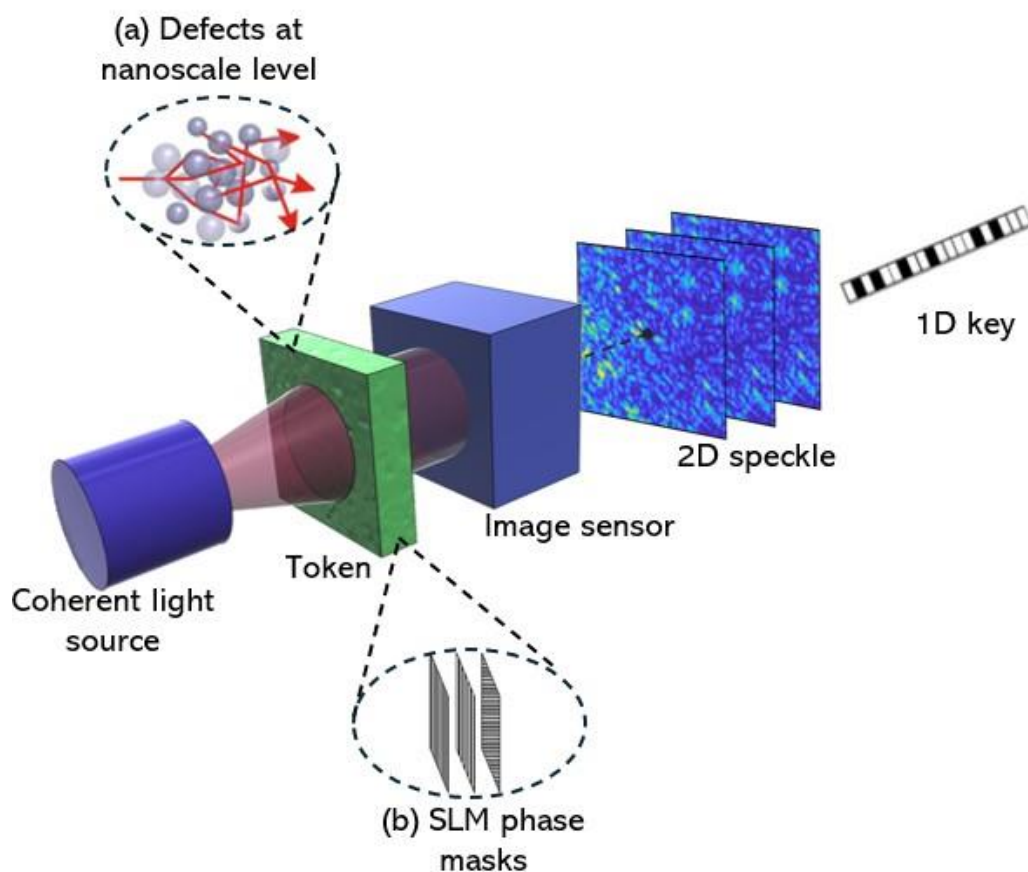


Figure 2.5: Schematic representation displaying the operating principle of a speckle imaging-based optical PUF when a typical disordered medium is utilized (a) and when phase masks generated by an Spatial Light Modulator(SLM) (b), are utilized as PUF “tokens”.

These refractive particles are in random positions, sizes, and orientations due to variations in the manu-

facture process. The light on the other side of this plate is detected in form of an optical speckle pattern (alternating and overlapping bright and dark spots) and analysed. Each response is determined by the optical scattering properties of the randomly disordered medium as well as the characteristics of the laser beam, such as its wavelength, spatial profile etc. The recorded speckle patterns (responses) undergo a hashing procedure resulting to unique bit-string outputs. While the physical characteristics of the o-PUF token are permanent in nature, the information extraction from PUFs (and other noisy sources like biometrics) is a probabilistic procedure; on a single challenge, a different response may be produced, due to the uncontrollable and random noise during the evaluation procedure (evaluation noise). To compensate the evaluation noise, the o-PUFs are usually combined with a fuzzy extraction algorithm that maps slightly different responses from the same challenge to the same unique output. In general, fuzzy extraction algorithm comprises two phases; the enrollment and the authentication. The former corresponds to the first time that a challenge is applied whereby the output string is generated along with a set of helper data, while the latter represents the noisy rerun of the enrollment phase, during which the same result is reproduced by using the helper data created.

Despite the unique characteristics of the typical disordered optical medium o-PUF approach described above, there are several challenges to be addressed towards an actual, cybersecurity-oriented implementation of a PUF instance of such a type, most of them associated with integration aspects and more specifically with the sensitivity to mechanical vibrations of the random medium, since even a displacement at micrometer scale may heavily disrupt the seamless operation of the device. Towards mitigating this constrain, in the context of ENTRUST, an alternative approach is explored, by replacing the medium with an SLM device, capable of generation on demand various phase masks to modulate the incident wavefront and thus the detected interference pattern (i.e. speckle). A schematic representation of the o-PUF instance considered here is depicted in Fig. 2.5(b). As for the key generation procedure in this case, the mechanisms and techniques applied in the typical o-PUF are also valid here.

2.4 Motivation in ENTRUST

In the context of ENTRUST, a TCB composed of a TEE and PUFs is envisioned providing a formidable solution for securing connected medical devices. Based on state-of the art analysis provided above as well as the detailed characteristics that each TCB trust anchor offers that will be described thoroughly in Chapter 3, it is evident that different types/levels of assurances are accommodated: i) hybrid-based or ii) pseudo-hardware- based(i.e.,TEEs) or iii) hardware-based guarantees (i.e.,PUFs) but for a single function. Thus, every security anchor has its own benefits providing also the capability of utilizing the most prominent of those depending on the host device-specific computational resources (high-end and low-end devices). The essence in ENTRUST project, since it revolves around CMDs, is the creation of an interoperable security stack that is able to operate on the top of different roots-of-trust.

PUFs can play a crucial role in generating unique device identifiers that serve as cryptographic keys for secure authentication whereas TEEs provide a secure execution environment for cryptographic operations, ensuring that sensitive data exchanged during the device onboarding process remains protected from unauthorized access. Once onboarded, connected medical devices rely on secure authentication mechanisms to establish trust within the network. Here, the combination of TEEs and PUFs enables robust device authentication procedures. PUFs generate device-specific keys that are securely stored within the TEE, ensuring that only authenticated devices can access sensitive resources and services. Additionally, TEEs facilitate secure communication protocols, encrypting data

transmissions between devices and backend servers to prevent eavesdropping and tampering. This ensures the integrity and confidentiality of medical data exchanged between connected devices.

Beyond authentication, maintaining the integrity and trustworthiness of connected medical devices requires continuous monitoring and attestation. Swarm attestation, enabled by the collective verification of device integrity within a network, can be facilitated using TEEs and PUFs. Each device periodically generates and attests to its integrity using its unique PUF-derived keys, with the collective results aggregated and verified by a centralized authority or swarm consensus algorithm. This dynamic attestation process ensures that any deviations from expected behavior, indicative of potential security threats or compromises, are promptly detected and mitigated. Thus, leveraging TEEs and PUFs for secure device onboarding, authentication, and remote attestation provides a robust foundation for safeguarding connected medical devices and preserving patient safety and privacy in healthcare ecosystems.

It has to be noted that SRAM-PUFs will be the main focal point here when it comes to the physical layer security, since they provide a CMOS compatible solution, they are cost effective compared to other silicon and non-silicon alternatives and they can be developed utilizing off-the-shelf SRAM chips. Their optical counterparts will be explored for research purposes, paving the way for future, elevated security CMD applications (e.g. quantum-resistive CMDs).

Chapter 3

The Customized ENTRUST TCB

After having introduced the two core components of the envisioned TCB of the ENTRUST framework, namely TEE and PUF, in this chapter we delve into the architectural details of the aforementioned TCB. Essentially, these cover all secure lifecycle management protocols and interfaces (that will be described in detail in the context of D4.2): from the secure device on-boarding and enrollment of the CMDs, and ENTRUST -related security components including the establishment of the necessary cryptographic primitives to the run-time monitoring and extraction of system measurements/properties, serving as trustworthiness evidence, and reaction policy enforcement mechanisms to any indication of risks and changes in the trust state of a device. All these operations are supported by ENTRUST 's customized TCB (CTCB).

This constitutes the palette of security mechanisms and ancillary processes that are required by all components offering the envisioned secure lifecycle management schemes. They are those hardware-, software- and firmware-based services, that are by default trusted, the combination of which is used towards the enforcement of a security policy. Part of ENTRUST 's TCB (as will be detailed in the chapters 4, 5 and 6) comprises the: (i) key management module for setting up and governing the use of all application- and security- related keys that need to be setup during the secure on-boarding of a device (ii) crypto primitives and protocols for continuous secure and authenticated communication as well as for attestation enablers (iii) tracing capabilities, to be exposed for capturing the continuous extraction of an extended set of device characteristics in a verifiable manner. The TCB must always behave as expected, otherwise, there is a risk for the entire trust assessment process to get exploited. Therefore, the entire TCB software stack configuration is monitored, stored and continuously verified by the underlying Root of Trust (RoT) so as to make sure that it is not altered. As described in Section 2.1.4, the TCB is minimized and constitutes the trust anchor for all applications/services that are instantiated in secure environments and for which we want to guarantee high operational assurance.

3.1 ENTRUST CTCB Functionalities

As introduced in the previous chapter, the TCB considered in the context of ENTRUST will be composed of TEE enclaves and a network of PUFs to establish a robust security framework tailored to the needs of existing CMDs. TEE enclaves will ensure the isolated execution of complex and safety-critical functions, safeguarding sensitive operations from potential threats. Concurrently, the network of PUFs will support various functionalities crucial for secure medical device operations. These functionalities encompass secure boot procedures, where the CTCB will mitigate diverse threats identified in prior of

the risk assessment. Furthermore, runtime attestation of software functions will be conducted, employing instruction-level code encryption coupled with verification of control-flow graph integrity and correctness. Additionally, the CTCB will facilitate secure data access and verifiable computing for health and patient signal monitoring, ensuring data integrity and confidentiality. Moreover, it will manage device identities and establish secure communication channels, enhancing overall device security and interoperability. The CTCB's adaptability enables the generation of verifiable chains of trust, utilizing attestation enablers such as proof attestations. These attestations provide assurance regarding the correct state and operation of the medical device in response to specific security claims. By doing so, the CTCB not only assures operational integrity but also enhances data confidentiality and integrity, fostering a secure environment for medical device interactions as well as data storage and exchange.

3.2 Trusted Computing Base Building Blocks

3.2.1 ENTRUST TEE Design Choice

The ARM Trustzone architecture, as described in subsection 2.2.2, provides a TEE implementation that enables running trusted applications in a secure and isolated execution environment. However, a major challenge exists for developers who write trusted applications with regards to the complexity of the platforms and system software. Specifically, while standards and APIs exist for traditional applications, e.g., POSIX, trusted applications are OS-specific, due to the use of custom libraries that are provided by different platform vendors. In turn, these libraries rely on drivers and firmware shipped with the platform by the manufacturer. Moreover, dispatching the trusted OSs requires specific firmware support, which increases the complexity. Thus, the portability of trusted applications across different trusted OSs and platforms is greatly hindered. Since the trusted OSs logic is common according to the Trustzone architecture, different frameworks exist to overcome this technical challenge. The most mature and common frameworks include OP-TEE and Trusty.

In the context of ENTRUST, special focus is given on OP-TEE, which is arguably the more mature open-source framework for developing trusted applications with ARM's TrustZone extensions in general purpose embedded systems. OP-TEE is a TEE implementation of GlobalPlatform specifications on top of TrustZone. OP-TEE provides a secure OS (running in the secure world) that is used alongside any generic Linux-based distribution running in the Rich Execution Environment (REE), also known as the normal world in Trustzone. To the best of our knowledge, unlike OP-TEE, Trusty mainly supports Android-based systems. Thus, to limit the technical efforts to integrate TEE with the use cases platforms and services that use various Linux-based distributions we opt to focus on the OP-TEE framework. Moreover, OP-TEE provides support for many off-the-shelf platforms, which is expected to further simplify the integration effort.

With OP-TEE, developers can write Trusted Applications (TAs) to run secure functionalities. Furthermore, TAs can be directly integrated into OP-TEE as Pseudo-TAs, which run inside the OP-TEE OS as secure privileged-level services. To simplify deployment, OP-TEE uses a generic build system, buildroot. Buildroot is a popular tool to deploy embedded Linux systems through cross-compilation. However, as buildroot is intended to be used by embedded systems, it is minimal by default. Therefore, adding support to different services requires manually modifying the build process and explicitly adding support for the required services.

Here, OP-TEE is utilized for enhancing the security posture of a interconnected healthcare ecosystem, protecting patient data and device functionality. Medical devices often handle highly sensitive information and perform critical functions, making them prime targets for cyber-attacks. By integrating OP-TEE into the device's software architecture, manufacturers can establish a trusted execution environment where sensitive operations such as encryption, decryption, authentication, and key management can be performed securely.

One of the primary ways OP-TEE can safeguard connected medical devices is through secure data storage. OP-TEE provides a trusted execution environment where sensitive data can be stored encrypted and isolated from the main operating system and other applications running on the device. This ensures that even if the device is compromised or attacked, the data remains protected within the secure enclave provided by OP-TEE. Furthermore, OP-TEE can facilitate secure data processing and communication within connected medical devices. By running critical operations within the trusted execution environment, OP-TEE can prevent unauthorized access to sensitive algorithms and cryptographic keys, mitigating risks associated with malicious attacks, software vulnerabilities, or unauthorized access attempts. Furthermore, OP-TEE facilitates secure communication between the medical device and external systems such as healthcare networks. By establishing secure channels for data transmission and implementing cryptographic protocols within the trusted execution environment, OP-TEE ensures the confidentiality, integrity, and authenticity of the data exchanged between medical devices and external entities.

Secure storage, as stated above, is a crucial function for the device operation of the CMD. In OP-TEE, such a functionality is implemented according to what has been defined in GlobalPlatform's TEE Internal Core API (also called Trusted Storage). This specification mandates that it should be possible to store general-purpose data and key material that guarantees confidentiality and integrity of the data stored and the atomicity of the operations that modify the storage (atomicity here means that either the entire operation completes successfully or no write is done). In the figure below (Figure 3.1), the secure storage system architecture that OP-TEE employs is presented. More details about the aforementioned architecture can be found in <https://optee.readthedocs.io/en/latest/architecture>

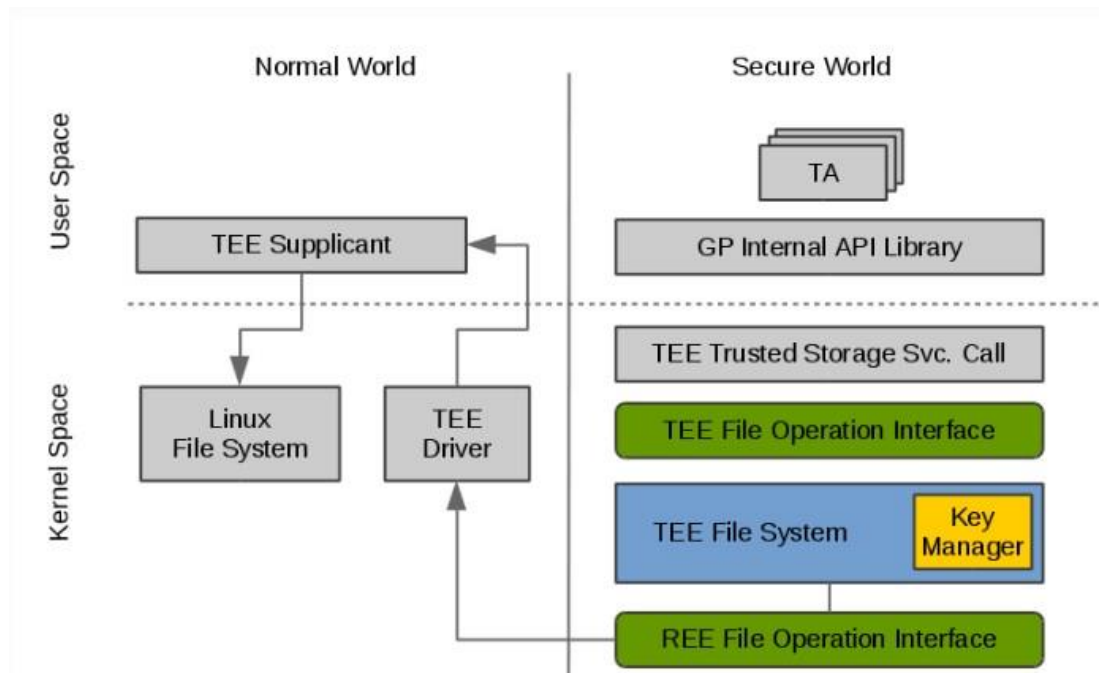


Figure 3.1: Secure Storage System Architecture of OP-TEE

3.2.2 SRAM-PUFs in the context of ENTRUST CTCB

In chapter 2, an introduction of PUFs was performed along with their unique inherent properties that will be utilized in the context of ENTRUST. Here, the specific requirements and characteristics that a SRAM module should exhibit in order to be considered as a PUF component will be elaborated whereas the evaluation results of the PUF instance considered will be presented. It has to be noted that in D4.1 the main areas of focus will be the SRAM-based PUF instance and how its unique properties can be leveraged in order to safeguard resources-constrained CMD. However, the high-level operational aspects and features of such an instance are also valid for the other PUF flavour that will be investigated during the project's lifetime, i.e. the speckle-imaging-based optical PUF, and will be documented in details in D4.3.

3.2.2.1 Design Crucial Characteristics of an SRAM-PUF

- *PUF properties determination and evaluation metrics*

In order to use a PUF as a solution to to obtain identifiers and cryptographic keys, its implementation-specific output (response) should fulfill some statistical requirements to ensure enough level of uniqueness (for security) and reproducibility (for reliability). These requirements can be established by means hamming distance. Hamming distance itself is the number of positions at which the corresponding symbols are different on two equal length strings. There are two types of hamming distance utilized, intra-class and inter-class hamming distance. Inter-class hamming distance is the distance between two responses resulting from applying the same challenge to two distinct PUF devices [62]. Intra-chip hamming distance refers to the difference between the two responses resulting from applying a challenge twice to a PUF device [23]. To ease the identification purpose, fractional hamming distance is also introduced. Fractional hamming distance is the number of differences between two strings divided by the length of the bit strings. In an ideal PUF, the intra-class fractional hamming distance (HD_{intra}), which is a measure of the PUF's **robustness**, has to be 0%, whereas inter-class fractional hamming distance, (HD_{inter}), which is a measure of the PUF's resistance to cloning (**unclonability**), has to be 50%.

One common application of PUFs is to use them to securely generate secret values (e.g., cryptographic keys). Such applications implicitly require that the adversary cannot predict the output of a PUF instance. Moreover, for typical PUF-based challenge-response identification protocols, it is important that the adversary cannot predict the response to a new challenge from previously observed challenge-response pairs. Therefore, the notion of **unpredictability** is an important property that needs to be included when applications of the aforementioned type are considered. Unpredictability is defined as the conditional probability of generating an identical output by applying to the same PUF component two different challenges [30]. In an ideal PUF system this probability has to equal to zero, thus the FHD between two PUF responses when two different challenges are applied has to be 50%. The derived HD distributions reflecting the characteristics of a PUF implementation in terms of robustness, unclonability and unpredictability, are usually visualized in the form of histograms in pairs. The main reason for these pairs being the verification of potential overlaps between them. If such overlaps exists, "false positive" or "false negative" conditions should be taken into account during the design phase of the PUF-based

instance. The former condition occurs when the system by mistake authenticates a PUF module as another one with characteristics. The latter condition happens when the system fails to authenticate the PUF units which are already recorded into the system correctly. It may be because of the noisy output. Choosing the number of bits that could be tolerated for errors by the system is a very challenging and tricky operation. This threshold value cannot be very big to reduce the false positivity and cannot be very small to handle the false negativity. The threshold is also different among PUF types, and it depends on the separation between the Intra-HD and the Inter-HD histograms. If both the histograms are not overlapping, then the threshold could be anywhere in the space between them.

In this case, the optimal value is in the middle of the valid threshold range as illustrated in Figure 3.2a. When overlaps between histograms occurs, the optimal threshold value is exactly at the intersection point. The overlapping region can be further categorized as false-acceptance rate (FAR) and false-rejection rate (FRR). The former lies in the intersection area of false positive, whereas the latter occupies the false negative area as depicted in Figure 3.2b.

• Helper Data Algorithms and Fuzzy Extractor

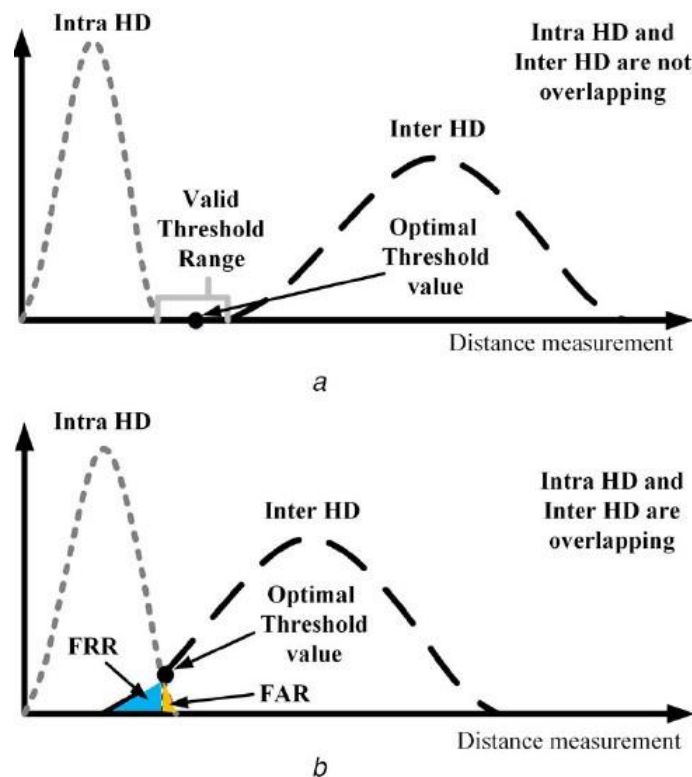


Figure 3.2: PUF-based system identification threshold for Intra and Inter (a) Non-overlapping, (b) Overlapping

There are two issues if PUF raw responses are used as a key in cryptographic primitive. First, both weak and strong PUFs rely on analog physical properties of the fabricated circuit to derive secret information. Naturally, these analog properties have noise and variability associated with them. This can be a problem due to sensitivity of cryptographic functions on noises of their inputs. Another issue is the PUF raw responses usually are not uniformly distributed, which makes it unqualified as a cryptographically secure key. These two issues can be solved using **Helper Data Algorithm (HDA)**. One can also refer Helper Data Algorithm as *fuzzy extractor* since both are capable of converting noisy information into keys usable for

any cryptographic application.

Fuzzy extractor solves both issues mentioned before by using two phases, *information reconciliation* and *privacy amplification*. In information reconciliation phase, possible bit errors are corrected to form a robust bit string [47]. Information reconciliation is tightly related to error correction. In fact, a procedure to do information reconciliation based on error correcting codes is called code-offset technique [27]. Using code-offset technique, one should be able to reconstruct a bit string w from a noisy version w' as long as the Hamming distance between w and w' is limited to t , where t is the maximum error correcting capability of the error correcting codes. The second phase, privacy amplification, is a process to evolve this robust bit string into a full entropy key. Privacy amplification, also can be called as randomness extraction [27], can be done by utilizing two-way hash function.

Beside these two phases, fuzzy extractor also consists of two procedures, **key generation** and **key reproduction**, as depicted in Figure 3.3. Generation (upper case of Figure 3.3) is a probabilistic procedure in which PUF generates a response w , according to the applied challenge, longer than the key and a hash value of the w is registered as the initial key. Concurrently, the helper data are calculated as $w \oplus c$, where c is an encoded random number sequence. Reproduction (lower case of Figure 3.3) is a deterministic operation, capable of recovering registered key leveraging the helper data created during the generation phase. The noisy response on the PUF to the same applied challenge w' is XORed with the helper data (i.e. $w' \oplus (w \oplus c)$) generating an error component c' . Then, the result of c' decoding is XORed again with the helper data and the initial key is generated following a hashing procedure. Therefore, a fuzzy extractor can be utilized to correct the errors of the noisy PUF responses and reproduce the initial key successfully, given that the Hamming distance between w and w' is limited to t .

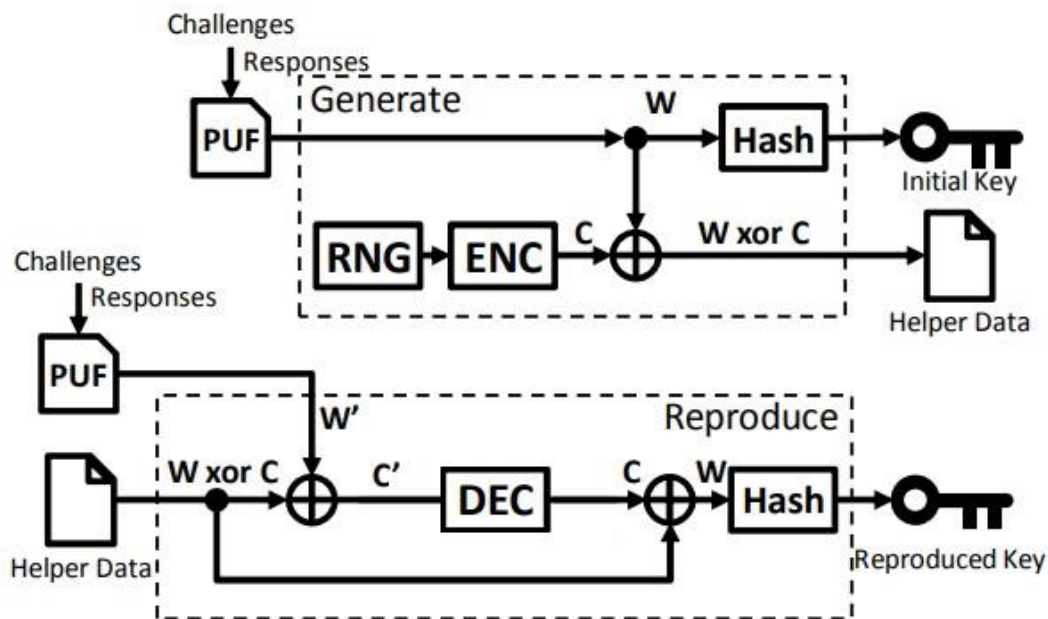


Figure 3.3: Generation and reproduction procedure of Fuzzy Extractor

• Error Correcting Codes

To handle the noisy response of a PUF instance, the utilization of error-correcting codes (ECC) is unavoidable. Error-correcting codes are a class of schemes for encoding messages in an attempt to

enable message recovery when noise introduced during the transmission or reception of the message [50]. ECC can be divided into two subcategories, hard-decision, and soft-decision. Hard-decision works on a predetermined set of values (usually 0 or 1 in a binary code), while a soft-decision decoder may take inputs on a span of values in-between (usually refers to float value).

One of the popular hard-decision error correcting code is BCH codes. BCH, stands for Bose-Chaudhuri-Hocquenghem, codes are a family of cyclic error correcting codes which constructed using polynomials over a finite field and work in a binary field. BCH codes are a very flexible set of codes in that within certain bounds there is a great amount of choice in code parameters and are relatively efficient in message length and error correction. Both BCH codes and Reed-Solomon codes have the capability to correct multiple errors. Reed-Solomon codes are also a flexible ECC and have similar parameters as BCH codes. Unlike BCH codes, Reed-Solomon codes can work in both binary and non-binary fields. Reed-Solomon codes also perform better in correcting burst errors while BCH codes are better at fixing random errors. BCH codes have an advantage where it requires less computing resource when working on the same parameter compared to Reed-Solomon codes.

3.2.2.2 SRAM module prerequisites for a PUF instance

As mentioned in subsection 2.3.3, an SRAM PUF utilizes existing blocks of an SRAM module to generate chip-specific data. Normally, when using SRAM for data storage, a positive feedback is given to force the cell into one of the two states (a '1' or a '0') available. Once it is there, the cell will be stable and prevented from transitioning out of this state accidentally. An SRAM module can be employed as a PUF by leveraging its start-up values. After powering up the circuit, each cell stabilizes at a state which is defined by the mismatches between the involved transistors and provides one bit of output data. Since this mismatch determines the value of the power-up state of an SRAM cell, the power-up state of a cell will be biased towards 0 or 1 depending on the mismatch value. Since all SRAM cells have been affected by random process mismatches during manufacturing, these start-up SRAM values can be utilized to generate a unique fingerprint.

To be eligible as a PUF component, an SRAM chip should:

1. exhibit stable outputs, which means that any noise must have minimal effect not only on its start-up behavior but also during its operation. Thus, it is crucial to evaluate the potential SRAM module that will compose the PUF instance during the design phase of the system, in terms of robustness under various conditions, since only elevated robustness can result to the enhanced security features.
2. generate logic '1' and logic '0' randomly with the probability of 0.5, ideally. This make a binary output string read from an SRAM array unique, random and non-traceable. The distribution of 1's and 0's can also be referred as *hamming weight*.
3. be capable of generating device-unique responses given the same challenge. In other words, the difference between the responses from different chips given the same challenge should be large enough to show that each SRAM is unique (well-separated HD_{inter} and HD_{intra}).

3.2.2.3 The SRAM cell and Bit selection algorithms

As presented in subsection 2.3.3, a typical six transistor CMOS SRAM cell (Figure 2.4) design utilizes

the concept of cross-coupled inverters, constructed by two inverters, each established by two transistors; inverter 1 by M_1 and M_2 , inverter 2 by M_3 and M_4 . Using such a design implies that the input of an inverter is the output of the other and vice-versa, which also indicates that the output of one inverter is exactly the opposite of the other inverter. Transistors M_5 and M_6 , referred as the access transistors, are used as the entry gate to the cell every time a read or write operation will be performed. The bitline (BL), the compliment bitline (\overline{BL}) and the wordline (WL) are employed as an entry to the cell. In addition, an SRAM cell will lose its state shortly after power down.

During manufacturing, there are small differences between each SRAM cell due to process variation which leads to a mismatch in the cell, which also means that the two inverters will always behave distinctly. The mismatch itself does not disturb the normal storage functionality of SRAM cell. Based on this bias, SRAM cells can be classified into three categories as shown below [23]:

1. Non-skewed cell: A non-skewed cell has no preference during its startup due to the impact of process variations does not cause any mismatch between the two inverters. This cell has a heavily fluctuated start-up value depending upon the noise introduced in the system
2. Partially-skewed cell: A partially-skewed cell has a small mismatch between the inverters which lead to a preference over value '0' or '1' but the cell can flip its value upon variation in external parameters
3. Fully-skewed cell: A fully-skewed cell is a heavily mismatched SRAM cell in a way that the cell inclined towards value '1' or '0' and has a resistance against external influence/noises.

Ideally, the cell of an SRAM-based PUF should fall into the fully-skewed class, since this type of cell guarantees that the PUF response of a given challenge will have small or no impact on its performance even under the presence of noise, which in turn leads to unstable bits flipping. The main sources of noise in SRAM cells are (a) power supply-induced voltage fluctuations, (b) ambient or voltage fluctuations-induced temperature variations, (c) transistors cross-talk due to the high integration density of SRAM cells and finally (d) aging due to the unavoidable transistor degradation over time.

The very first step for an SRAM module utilization as a PUF, is the determination of the challenge-response pair pool. In SRAM PUF, there are two types of challenges that can be applied to the system. The challenge can be either the whole SRAM memory or specific addresses. If a set of addresses is given as a challenge, an address in there can refer to an address of a byte, a bit, or a sequence of bytes or bits. If specific addresses of SRAM cells are used for PUF challenge, one of the major steps on using SRAM PUF is the determinations of its stable bits. Stable bits themselves refer to fully skewed cells explained before. Even though the error correction code is present to correct the noise of bit responses, it also has a limitation on how many bits it can correct. Choosing the most stable bits is important to ensure that the PUF result is always the same throughout its lifetime. Below we present two known algorithms to search for stable bits:

- **Neighbor Analysis:** This algorithm is using the rank of total stable neighbors and has been which proposed by Xiao et. al [73]. They argue that the cells which are most stable across environmental conditions are surrounded by more stable cells during enrollment. A stable cell surrounded by more stable cells has a tendency to become more stable because its neighboring cells are likely to experience similar aging stress and operating conditions. In this algorithm, all the stable cells

are given weight according to the number of stable bits surrounding it. The more stable neighbor cells it has, the higher weight it gets. For example, if a cell is not stable, it is given zero as its score. If it is stable, at least it will get score one. If it only has one stable neighbor on each left and right side, it will get score two as result of an addition of one from being a stable cell and one from having a stable neighbor on both sides. To get score three, it needs to be stable and has two stable neighbors on left and right sides. After determining the weight of each cell, a heuristic algorithm that greedily chooses cells for the PUF ID/key with weight greater than a threshold is used. Before the algorithm is performed, one should collect lots of SRAM cells value first. The data should be retrieved in various condition, for example, different voltages, temperatures, and time differences between enrollment. Afterwards, using the data gathered, the location of all stable bits in SRAM need to be located. A stable bit has to has the same value in all enrollment. Last, the neighbor analysis algorithm is performed to get the most stable bits in SRAM.

- **Data Remanence Approach:** Another bit selection algorithm is by using the data remanence of SRAM cell [46]. There are only two remanence tests involved in this approach: first, writing a value (1 or 0) to the whole memory and second, briefly turning off the power until a few cells flip. The most robust cells are the cells which effortlessly flipped when written with the opposite data. Strong 1's are bits that are flipped fast after 0 is written to its location. On the contrary, if 1 is written to a bit location and the bit flipped fast, it means that the bit is a strong 0. When using this approach, one should carefully determine the temporal power down time. On one hand, if the temporal power down period is too short, then the data will stay in the previously written state.

A significant advantage using this algorithm compared to the previous one is a much shorter time required to locate stable bits. Using neighbor analysis, there are many SRAM values need to be gathered first which might take hours or days. Locating stable bits from hundreds of data probably also take time as well. If data remanence approach is utilized, there is no need to gather many data. One only need to determine the temporal power down required to get strong bits required. Since usually the temporal down period required is less than 0.5 seconds, this analysis only takes few minutes.

3.2.2.4 ENTRUST's SRAM-PUF

Before we delve into a detailed description regarding the SRAM-PUF design implementation tailored to the needs of an CMDs ecosystem, a set of assumptions has to be defined:

- Only the manufacturer of the SRAM-PUF modality has physical access to the device
- An potential adversary may gain the knowledge of the helper data and challenge used in the PUF concept considered here
- The key-length security level is 256-bits
- The SRAM module utilized should be be easily available in the market and cost-effective

Considering the aforementioned assumptions, a solid methodology has been followed towards a reliable and secure SRAM-PUF instance. Its steps are summarized below:

1. The initial step for the composition of the ENTRUST SRAM-based PUF implementation was the **selection of the appropriate embedded platform to host the proposed system**. There are two

major candidates that should be considered here, Arduino and Raspberry Pi, due to their popularity (promoting debugging process), availability (can be found easily) and flexibility (various flavours released with diverse computational resources). Besides those three factors, another feature which lead on selecting either Raspberry Pi or Arduino, is the existence of a General-Purpose Input/Output (GPIO) interface. Such an interface is extremely vital for the electronic PUF proposed here since it fosters the seamless communication between the SRAM and the embedded platform. Compared to Arduino, Raspberry Pi offers a higher computing capability and relatively easier development, but Raspberry Pi requires a longer startup time compared to Arduino, whereas its power consumption is greater. Thus, considering also the requirements posed by the CMDs ecosystem per se, Arduino platform was chosen and more specifically the Mega 2560 variation. Arduino Mega 2560 offers larger memory capability compared to other types, such as 256k bytes of Flash memory, 8k bytes internal SRAM, and 4k byte EEPROM. Besides, it also has 54 digital I/O pins and 16 analog I/O pins which ease the interfacing to the external SRAM.

2. After choosing the proper host platform tailored to the needs of ENTRUST SRAM-based PUF implementation, the next step was the **selection of the SRAM module** considered as a root-of-trust here. To that end, Cypress CY62256NLL-70PC SRAM chip was utilized, which is a cost-effective, widely used SRAM module with 256 kbit capacity and an automatic power-down feature that reduces the power consumption by 99.9% when deselected. Then, the aforementioned module was connected with the Arduino Mega 2560 in order to evaluate its characteristics. One crucial step towards that direction was the calculation of 1's and 0's distribution throughout the whole SRAM cells. To do so, all SRAM bit locations was set as a challenge and the resulting sequence of bits was recorded for 5 distinct Cypress CY62256NLL-70PC SRAM modules. Analyzing the extracted bit-strings, the 1's and 0's distribution for all devices tested was well-balanced around 50%, showcasing that the selected SRAM exhibits a sufficient amount of randomness.

3. Since one of the most essential criterion for a PUF instance was fulfilled, the proposed device's **robustness, unpredictability and unclonability characteristics were determined in terms of FHD**. Prior that, by following the data remanence approach described in subsection 3.2.2.3, the positions of the stable and unstable bits of the SRAM were determined. The former can be employed as a challenge for device authentication schemes, whereas the latter can be utilized when the TRNG mode of operation of the PUF is selected. Here, for the evaluation purposes, the locations of the stable bits were used to probe the SRAM module. Considering that the number of the stable bits within the SRAM was found to be equal 246678 bits and assuming for instance that only 4662 stable bits will be needed to create a sting of 2331 bits long, the number of possible challenges can be calculated from the permutation of the required bits and the available bits, i.e. $P = \frac{(4662!)}{(4662-2331)!} \approx 8.97^{8240}$. Thus, based on such a poll of challenges, which correlates with a corresponding set of responses, the proposed SRAM-based system can be considered as a strong PUF.

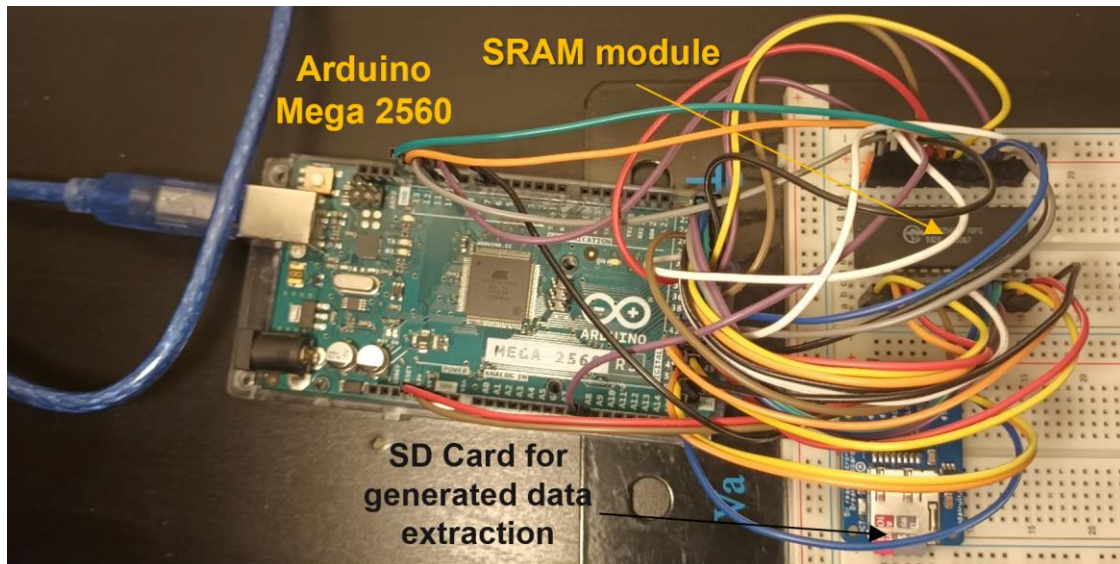


Figure 3.4: The experimental setup considered here for the characterization of the proposed SRAM-based PUF implementation.

In order to evaluate the operational characteristics of the SRAM-PUF, a set of 10,000 challenges were generated by scrabbling the locations of the SRAM stable bits, from each distinct SRAM chip. The first characteristic evaluated was the system's robustness. To do so, one challenge from the previously generated set of one SRAM (i.e. challenge No.1 of the SRAM No.4) was selected to probe the specific SRAM module connected to the Arduino Mega 2560 (Figure 3.4), whereas its response was recorded and saved in an external medium (SD card). This procedure was repeated every 15 seconds, for 96 hours consecutively, whereas periodically the SRAM module was switched off and turned on again to investigate also any potential effect in the device operation due to on/off switching. The same procedure was followed by utilizing the rest SRAM chips purchased. In the following figure (Figure 3.5), an indicative example of the results extracted in terms of FHD is depicted. As seen from the corresponding histogram (red bins), the FHD distribution from the specific SRAM module is centered around 2.14%, highlighting the stability of the proposed PUF, since the deviations between the generated bit strings by applying the same challenge, over a long time period of continuous operation, are minor. It has to be noted that all SRAM chips tested here exhibited similar behaviour in terms of robustness, ranging from 2.08% to 2.2%. Such an elevated feature will be extremely useful for the application domain that ENTRUST targets and will be leveraged towards the standalone SRAM-PUF realization, since it is directly correlated with the error correction capability that any PUF requires in order to reconstruct its noisy outputs. The selected level of the aforementioned capability is inevitably associated with the computational resources utilization and based on the fact that in general CMDs are considered constrained devices, exceptional robustness characteristics supports the overall efficiency of the device, which is one of the most crucial aspects that has to be considered here.

After the PUF's robustness evaluation, the next parameter investigated was the unpredictability. To perform that study, one SRAM chip was selected, loaded to the breadboard and connected to the Arduino Mega 2560. Then, the SRAM module specific set of the challenges generated previously, i.e. 10,000 challenges, was employed in order to initiate the bit sequence generation

procedure, which upon their creation were captured and analysed, again in terms of FHD. The results of the unpredictability measurements for the SRAM under test, are presented in Figure 3.5 (orange bins). As depicted, the FHD distribution is centered near 47.9%, which is very close to the ideal value, demonstrating the non-predictable nature of the implementation considered here, since the probability of generating an identical output from the specific SRAM-PUF instance when two different challenges are applied is almost equal to 0. Thus, this is another indication the SRAM chips considered here are capable of generating bit sequences of high-entropy.

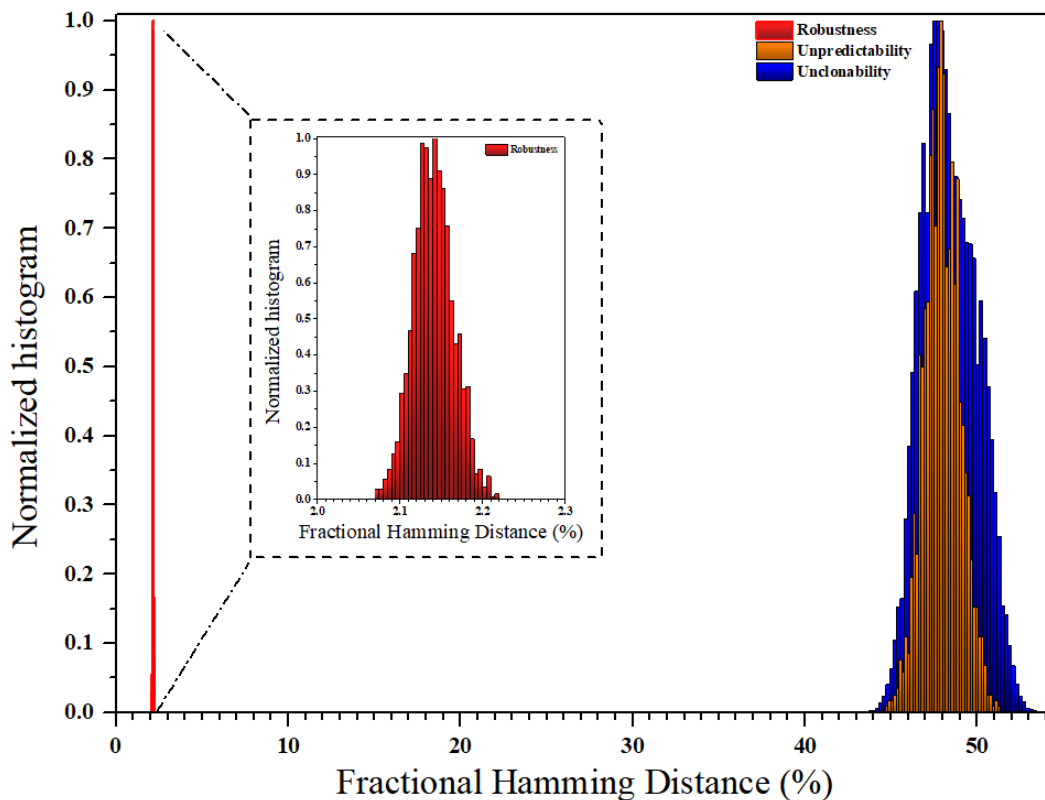


Figure 3.5: The SRAM modules' crucial characteristics in terms of FHD.

Finally, in order to evaluate the unclonability of ENTRUST SRAM-PUF instance, one set of challenges extracted from one SRAM chip (e.g. the 10,000 challenges of SRAM No.2) was utilized in order to probe the rest of the modules. Each SRAM module was loaded and connected with the Arduino Mega 2560 and upon the application of the aforementioned challenges the responses of each SRAM were captured and analysed. The results of such an analysis, in terms of FHD, are presented in Figure 3.5 (blue bins). As depicted, the FHD distribution is centered around 48.26%, which is again very close to the ideal value, whereas robustness (intra-class) and unclonability (inter-class) distribution are well separated, without the presence of overlaps between them, highlighting that the creation of an identical clone of such a module is almost not feasible even by the manufacturer per se, whereas the false acceptance or rejection rate is equal to 0.

4. Upon the selection of the proper host platform for the needs of ENTRUST SRAM-based PUF implementation as well as the evaluation of most important SRAM modules characteristics, **the type of error-correcting codes which will be employed as an element in the key generation scheme was selected.** One vital consideration for such a choice is the available memory of the

embedded platform since the error correcting codes has to fit in the embedded platform's internal memory. From the two options mentioned in subsection 3.2.2.1 (i.e. BCH and Reed-Solomon), BCH codes were particularly picked as the ECC due to lower computing resource required compared to Reed-Solomon Codes. BCH also better at correcting random errors than Reed-Solomon Codes. BCH codes are flexible ECC shown by multiple parameters available. The only fixed parameter is q since the problem is in binary form ($q = 2$). The source code for BCH codes utilized in the context of ENTRUST is a modified version of Robert Morelos-Zaragoza's version which can be retrieved at <http://www.eccpage.com>. This code is selected since it can support m ranging from 2-20 which mean the length of the code that can be corrected is ranging from 2 until 1048575 (m parameter is the power to which q has to be raised in order to generate a Galois Field for the construction of the code.) When using BCH codes, one should be careful on deciding the parameters that will be used, since for example, high m values means elevated memory needs. Considering that the internal SRAM capacity of the Arduino Mega 2560 is 8k bytes capacity as well as to maximize the error correction capability, the resulting BHC parameters utilized here are: $n=63$, $k=7$, $d=31$, $t=15$, where n is the block length of the code, k is the number of message bits in a codeword, d is the minimum designed Hamming distance between distinct codewords, and t is number of errors that can be corrected, respectively. Such a combination of BCH parameters will enable an error correction capability of 23.8% of the generated bit-string. It has to be noted that the aforementioned set of BCH parameters was selected for the needs of the first round of experiments and may be modified if needed for elevated efficiency purposes.

5. One of the most crucial steps towards a reliable SRAM-based PUF implementation that can be utilized as a trust anchor in the context of ENTRUST ecosystem was the **design of the key generation and reconstruction scheme**. In the context of ENTRUST, the aforementioned scheme will be a modified version proposed in [38] and more details will be provided in D4.2, accompanied by a comprehensive description of the security protocol employed for the secure enrolment and authentication of a CMD.

3.3 High-Level Description over the ENTRUST's TCB Security Architecture

In the previous subsection, a comprehensive description regarding the components that will constitute the trust anchors of the envisioned TCB was provided along with all the associated specifics and features that they possess. Here, the security architecture of the ENTRUST TCB towards enabling the trust establishment within CMDs ecosystem, is defined for two distinct types of devices: i) those with elevated computational resources (high-end devices) and ii) their constrained counterparts (low-end devices). In D4.2, a comprehensive description along with the detailed ENTRUST's TCB architecture and its components will be provided along with all the relevant operations/processes that its trust anchors offer/support, for both types of devices that are essential parts of the ENTRUST ecosystem.

3.3.1 High-End CMDs TCB Overview

In Figure 3.6, an overview of the ENTRUST's security stack is provided, where CMDs with elevated computational resources are considered. Components in green offer extended interoperability across

heterogeneous architectures, providing a unified level of abstraction and enriching existing security features, whereas special focus is given to the **trusted services** that can be supported by the envisioned interoperable security stack towards a secure device lifecycle management, including: i) **Trust assessment**, ii) **Secure Device Onboarding** and iii) **SW updates**.

Verifiable Credentials Management

In the context of ENTRUST, a set of trust-aware continuous authentication and authorisation processes operating in tandem with the secure service onboarding process are provided. The authentication and the onboarding of devices requires evidence on the correct state of the device. That is, the attestation outcome is treated as a security claim which is transformed into a verifiable presentation using the Verifiable Credentials Management enclave in the TEE. The security claims, which prove whether a device is in a correct state or not, are used during the onboarding or any interaction with the blockchain and are utilized as enablers for data source integrity verification and as attributes for attribute-based access control. This ENTRUST functionality implies that only trusted entities will take part in the ENTRUST ecosystem. That is, the Verifiable Credentials Management is considered part of the ENTRUST TCB.

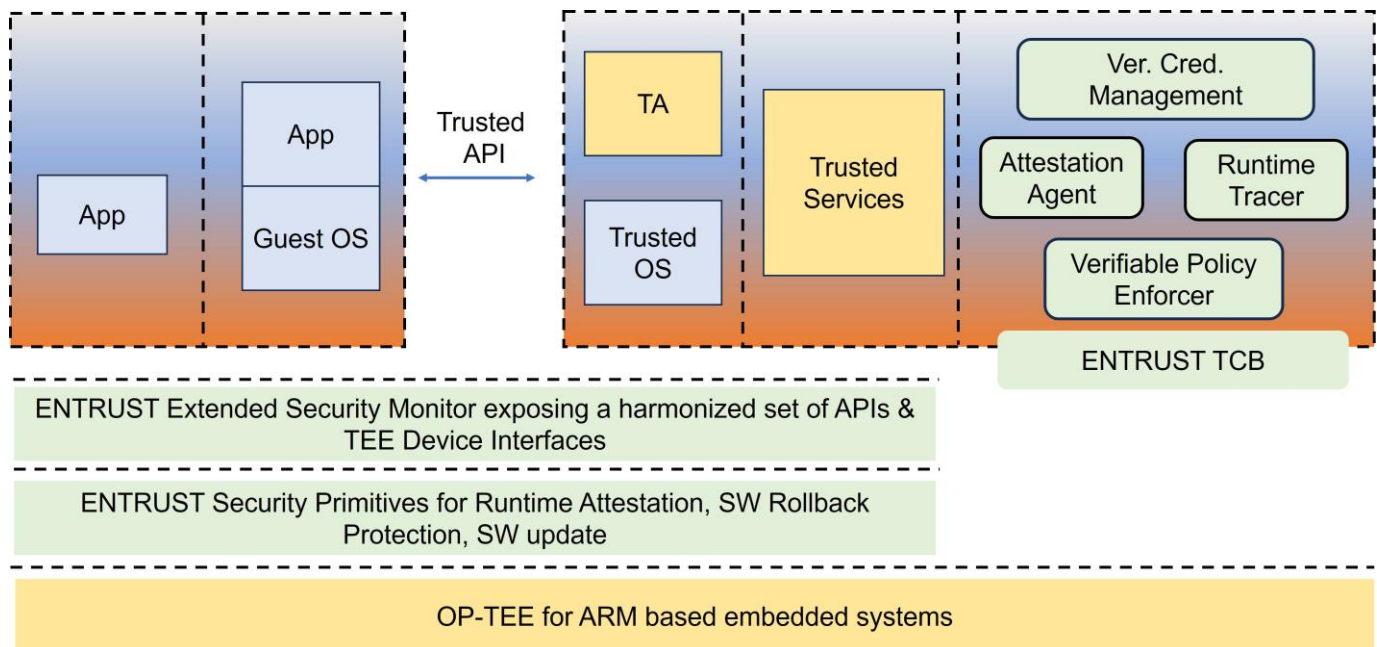


Figure 3.6: ENTRUST security stack for high-end CMDs.

Attestation Agent

Attestation is a security service in order to validate the integrity of an entity such as an CMD or an installed software. Static attestation mechanisms enable the detection of manipulation of a device's static memory content (e.g., the program code or program's configurations) whereas runtime attestation mechanisms provide information about a device's runtime behaviour, allowing the verifier to detect also dynamic attacks. ENTRUST utilizes attestation for verification of CMDs' operational assurance and the attestation Agent is the technical component of that performs the aforementioned task. It exposes TEE Device Interfaces responsible for providing the run-time system measurements capturing the current device's configuration and operational state, as obtained from the Tracer. Given the traces, the attestation

agent can infer the trustworthiness of a service.

Runtime Tracer

ENTRUST's runtime tracer is responsible for continuously monitoring processes and routines that are executed in the untrusted world of each device. Its primary scope is the collection of essential information for attestation methods employed in ENTRUST, for ensuring integrity. The tracer, in essence, is capturing hashes of configuration properties from untrusted processes and routines. Runtime tracer is reported as part of the TCB, referring to the piece of the tracer that runs in the trusted world. In fact, all the security critical functionalities of the tracer are hosted within the trusted world. More specifically, the untrusted part of the Tracer is responsible for fetching the raw traces of the attested application, whereas the trusted part of the Tracer is responsible for decoding the raw traces, in order to calculate the runtime configuration hash, and for supporting the generation of the digital signature over the configuration hash based on the secret key, are taking place.

Verifiable Policy Enforcer (VPE)

Acts as the entity responsible for validating the current liveness of the Key Restriction Policy Engine (KRPE) being enforced. Its primary function is to prevent potential attackers from having multiple obsolete policies active. For instance, if a past version of an application is associated with a specific policy with known vulnerabilities, the VPE ensures that this policy is identified and characterised as obsolete, preventing unauthorised or outdated policies from influencing the system's security posture and allowing an attacker to exploit them. The VPE monitors the correctness (i.e., integrity) of the software versions executed by the runtime tracer and the attestation agent (i.e., parts of TCB) within the enclave, and if the versions are successfully verified, then the VPE authorises the enforcement of a specific key restriction usage policy. For the operations needed by the VPE, the enclave will hold a specific cryptographic key. The VPE component needs this key to sign the accepted and verified software versions of the attestation agent, the Tracer and any other security critical application that is running within an enclave is the expected one, as a defense against denial of software updates.

3.3.2 Low-End CMDs TCB Overview

As mentioned previously, the essence of the ENTRUST's TCB is to provide a dynamic adjustable technical solution tailored to the specific needs and capabilities of CMDs. Having provided an overview regarding the TCB of high-end CMDs, in Figure 3.7 the corresponding security stack for their resources-constrained counterparts is depicted.

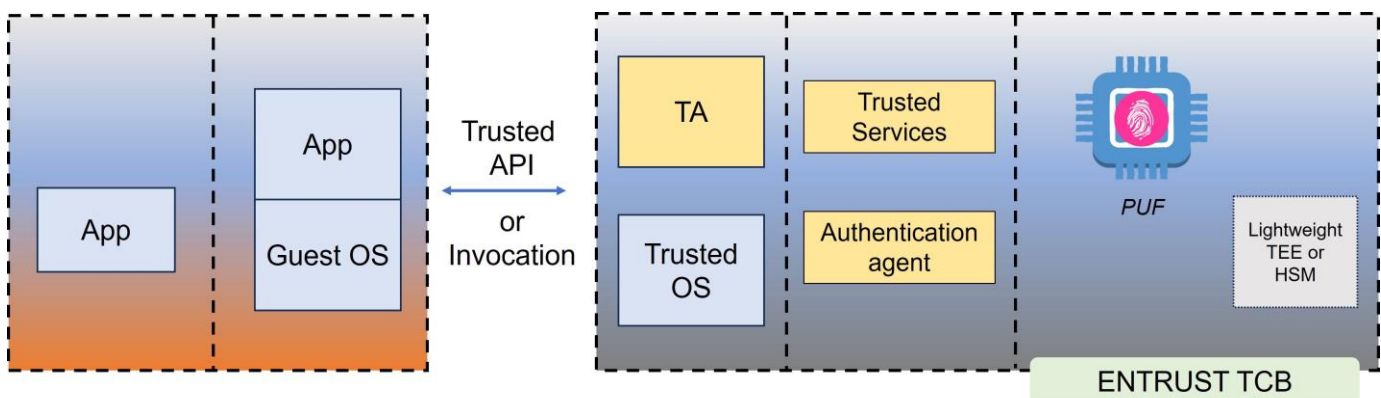


Figure 3.7: ENTRUST security stack for low-end CMDs.

In this case, the building blocks of the low-end CMD TCB are not operating within enclaves and the TCB is composed mainly from a PUF instance and a lightweight TEE or HSM serving only for enabling the (run-time) authentic tracing of the device's state, providing the necessary system configuration measurements as well as in-device state transitions. The trusted services here, are strongly correlated with the main functionalities of the PUF, with are the **key generation and reconstruction**, whereas an authentication agent was included here, in order to provide continuous authentication between the low-end and the high-end devices. More details about the exact schemes that will be followed regarding the PUF-assisted key generation and reconstruction as well as the aforementioned agent will be provided in D4.2.

3.3.3 Engineering Stories Supported by ENTRUST's Dynamically Adjustable to the type of the device TCB

In this subsection, a detailed mapping of the functional specifications that will be expressed as engineering stories in the following chapters, is provided in Figure 3.8. It has to noted that modifications may apply upon the initiation of the first implementation phase and will be reported, if needed in D6.1.

Engineering Stories supported by ENTRUST TCB		
Engineering Stories	TCB of high-end CMDs	TCB of low-end CMDs
Story-I	✓	
Story-II	✓	✓
Story-III	✓	✓
Story-IV		
Story-V	✓	
Story-VI	✓	
Story-VII		✓
Story-VIII		
Story-IX		
Story-X	✓	✓
Story-XI	✓	✓
Story-XII	✓	✓
Story-XIII		
Story-XIV		
Story-XV		
Story-XVI		
Story-XVII	✓	
Story-XVIII		
Story-XIX		✓
Story-XX	✓	✓

Figure 3.8: Engineering stories supported by ENTRUST's customized TCB mapping

Chapter 4

Cryptographic Functionalities for Supporting the Secure Lifecycle of CMDs

As we have discussed in D2.1, and we will fully clarify in D4.2, regarding the security functionalities inherent to the ENTRUST platform, our overarching objective is to furnish a robust solution for the secure lifecycle management of CMDs, spanning from the booting and onboarding processes within a service graph chain to the need for secure software updates and remote attestation. This holistic approach encompasses an array of cryptographic primitives and trust extensions essential for both static and runtime operational assurance, facilitating the measurement of a CMD's trustworthiness level.

Integral to ENTRUST's mission are the security requirements underpinning its functionalities, centering on the utilization of cryptographic primitives and building blocks for robust authentication, trust-related evidence sharing in a privacy-preserving manner, and the creation of conformity certificates. These certificates, intricately linked to access control policies, ensure stakeholders possess an updated understanding of a device's trust state while safeguarding sensitive information against disclosure attacks.

ENTRUST adopts the IETF standardized protocol of Manufacturer Usage Descriptions (MUDs), augmenting it with eMUDs containing policy hashes governing cryptographic key usage. This enhancement enables the platform to facilitate new mechanisms for the exchange of security-related evidence in a privacy-preserving manner, thus bolstering its efficacy in safeguarding CMDs.

Given ENTRUST's emphasis on secure lifecycle management, the establishment of a protected set of keys constituting a device's key hierarchy becomes imperative. This hierarchy necessitates a harmonized management system to efficiently manage interlinked keys of various types, optimizing security and operational efficiency.

Attributes within ENTRUST extend beyond mere identity attributes to encompass a comprehensive assessment of CMD trust levels during runtime. Trust considerations encompass properties ranging from integrity to safety, resilience, and robustness, assessed based on evidence collected and monitored during runtime. Each trust property is associated with an Attribute Key, facilitating trust assessment based on collected evidence and ensuring transparency across stakeholders. For instance, the identity of a device or the fact that the CMD has a TCB or the capability of secure boot are attributes that are associated with their Attribute Keys. All those attributes represent different trust characteristics based on which the trust assessment can be conducted they are depicted as attribute keys. Because in ENTRUST, these types of attributes will be required by any entity/stakeholder to be disclosed.



The envisioned entities within the ENTRUST platform, each with distinct security requirements, play integral roles in its operational ecosystem. From the Manufacturer Domain, involving manufacturers and AAA servers, to the Domain Manager orchestrating service graph chains, and the Service Provider entities like hospitals, to the Security Administrator overseeing integration and maintenance, we strive to capture the diverse security needs of these entities through Engineering Stories presented in this deliverable. This comprehensive approach ensures the alignment of ENTRUST's functionalities with the security requirements of its operational domains, thereby enhancing its efficacy in safeguarding CMDs.

4.1 ENTRUST Cryptographic Toolkit

The ENTRUST framework will be offering a set of crypto-based operations starting from conventional signature and encryption schemes to Attribute-based Encryption (ABE) and Attribute-based Signatures (ABS) schemes. Also, it will explore some interesting attestation schemes such as Swarm and property-based attestations. ENTRUST will also seek suitable Zero Knowledge Proof and commitment schemes that support use case requirements. These crypto schemes aim to enhance blockchain-based operations for secure and auditable data sharing among stakeholders and support privacy when needed.

ENTRUST crypto schemes will be built on top of the fundamental cryptographic operations that are provided by the underlined trusted component. We mention some of these modules of the crypto toolkit:

Pseudo (P) and True (T) Random Number Generator (RNG): ENTRUST uses OP-TEE's PRNG module (high-end devices) and SRAM-PUFs TRNG mode of operation to generate values for nonces, key generation, and for randomness in signatures. A PRNG nominally consists of an entropy source and collector, a state register, and a mixing function (typically, an approved hash function). A PRNG may produce numbers that are indistinguishable from random with a uniformly chosen "small" input in a deterministic way while generating the same output for a specific input. A TRNG utilizes a physical entropy source to generate a random number sequence.

Hash Functions: A Hash function is a cryptographic function that takes a message of any length and compresses it to a digest of fixed length. For a hash function to be cryptographically secure, two important properties should be satisfied for a hash function to be secure:

1. One-way: It is computationally infeasible to find any input that maps to any pre-specified output.
2. Collision resistant: It is computationally infeasible to find any two different inputs that map to the same output.
3. Second pre-image resistant: It is computationally infeasible to find another input that produces the same output as an existing input.

Hash functions are used to provide integrity checking and authentication as well as one-way functions, as needed, e.g., in Key Derivation Functions (KDFs). There are several types of hash Algorithms such as SHA-1, SHA-256, and SHA-384.

An HMAC is a keyed hash function. It performs a secure hash operation using a shared secret key called the HMAC key. It assures that protected data was not modified and that it came from an entity with access to a key value. HMAC ensures the integrity of data that may be stored externally, it is a proof that an attacker has not altered the data value. HMACs can also be used to derive keys, using the so-called KDF.

Symmetric Encryption: Symmetric encryption is a type of encryption where only one key, called a secret symmetric encryption key, is used for encrypting and decrypting data. This encryption method differs from asymmetric encryption where a pair of different keys, one public and one private are used to encrypt and decrypt messages respectively. An example of a symmetric key algorithm that may be considered in the ENTRUST framework is the Advanced Encryption Standard (AES).

Asymmetric Encryption: Asymmetric encryption, also known as public key encryption, uses a public key-private key pairing: data encrypted with the public key can only be decrypted with the private key. ElGamal encryption system is an example of an asymmetric encryption algorithm.

Attribute-based Encryption (ABE): ENTRUST will adopt Attribute-based Encryption (ABE) that allows only entities that hold specific attributes to decrypt the encrypted medical data stored in the ENTRUST blockchain. ABE is an essential access control technique that regulates access to the original data in a way that allows users who hold the necessary attributes to decrypt the cipher text and retrieve the raw data. Attribute-based encryption (ABE), was first introduced by Sahai and Waters [35]. There are two kinds of ABE, ciphertext-policy attribute-based encryption (CP-ABE) and key policy attribute-based encryption (KP-ABE). In CP-ABE, a user encrypts the data according to a predicate (access policy) defined over attributes, such that only the party that possesses a secret key associated with the attribute set satisfying the predicate can decrypt the ciphertext. In KP-ABE, the encryptor has no control over who has access to the data he encrypts. In (KP-ABE) the private keys are associated with access structures that control which cipher texts a user can decrypt.

Digital Signatures: A Digital Signature is a cryptographic construction used for verifying the authenticity of data. A valid signature on a message gives a recipient confidence that the message (data that is being signed) came from a sender known to the recipient. Signing a message can be done using either an asymmetric or a symmetric algorithm. The method of signing depends on the type of key. For an asymmetric algorithm, the methods of signing are dependent on the algorithm (RSA or ECC). For symmetric signatures, only the HMAC signing scheme is currently defined.

Attribute-based Signatures (ABS): An ABS signature on a message and a predicate over the universe of attributes is a type of signature where the users sign messages with any predicate of their attributes issued from an attribute authority. An ABS attests not only to the identity of the individual who signed a message, but also to the fact that a user satisfies a given predicate/policy. ABS is useful in many important applications such as Blockchain access control, anonymous authentication, and attribute-based messaging systems.

4.2 State of the Art

4.2.1 Enhanced authorization protocol for medical device on-boarding

In recent years, the number of Internet of Things (IoT) devices has increased dramatically, becoming integral to various aspects of our daily lives, including transportation, industry, and healthcare. This proliferation has introduced new security challenges and expanded the surface for cyberattacks, exemplified by cases as the Mirai botnet [74]. At ENTRUST, we recognize these vulnerabilities,

especially in the healthcare sector, and are committed to enhancing the security of all connected healthcare devices to protect sensitive data and ensure patient safety [54]. The security of these devices hinges on effective identification and authentication, which depend on cryptographic techniques. However, the limited resources of IoT devices often necessitate the use of tiny cryptographic systems. These smaller systems can make the devices more vulnerable and easier to hack.

At ENTRUST, when a CMD is deployed inside the network, it is crucial to ensure proper authentication [68]. For this purpose, we utilize the Extensible Authentication Protocol (EAP) [5] solution, an authentication framework defined by the IETF [6].

EAP: The Extensible Authentication Protocol (EAP) is a network security framework that provides a flexible method for authenticating the identities of devices seeking to access a network and the network itself to those devices. Originally developed as an extension of the Point-to-Point Protocol (PPP), EAP supports a wide range of authentication methods such as token cards, Kerberos, One-Time Password (OTP), and Public Key Infrastructure (PKI) [22], making it highly adaptable to various security requirements and allow them all to work over the same network infrastructure.

Its operation is based on a client-server model, where the client (supplicant) requests access and the server (authentication server) coordinates the authentication process. This interaction often involves an intermediary, such as a router or access point, particularly in wireless networks. The server first requests the client's identity, and based on the response, it selects an appropriate EAP method (like EAP-TLS [67], PEAP [26], or EAP-TTLS [31]) to proceed with the authentication. These methods can provide strong security features, such as mutual authentication and encrypted data exchange.

Finally, its independence from the network transport layer allows it to function over both wired and wireless network infrastructures, making it versatile for various applications. It is particularly vital in setups that require robust security, such as enterprise Wi-Fi networks and internet-based connections, ensuring that only authorized devices can access network resources. This capability has made EAP integral to protocols like Wi-Fi Protected Access (WPA) and IEEE 802.1X, which are standards for securing Ethernet and wireless networks [59].

However, EAP is not designed to manage the dynamic properties of network connections. It does not address changes that may take place during a session, such as modifications to network policies or CMD privileges. This is where the Manufacturer Usage Description (MUD) [40] framework is employed. MUD is used to specify the intended use of devices, allowing network administrators to set appropriate access and security policies.

MUD: Manufacturer Usage Description is a standard that enhances security for IoT devices by enabling them to communicate their network requirements to the networks they connect to. This allows IoT device manufacturers to specify the types of network connections necessary for the devices to function properly.

This implementation is done through a MUD File, which is a JSON [41] document that describes the device's intended behaviors, such as the servers it should connect to and the types of traffic it should allow. This file is hosted by the manufacturer and retrieved by the network controller using a URL that the devices provides when it joins the network.

By using MUD, networks can automatically configure access controls that are tailored to each device's needs, significantly reducing the risk of inappropriate access or cyberattacks [37]. These setups help in minimizing the potential damage that a compromised device could cause to other parts of the network.

The integration of EAP and MUD profiles within ENTRUST's Domain Manager component aims to dynamically enhance network security. During the bootstrapping phase, the Domain Manager initiates an authentication process upon receiving the MUD-URL of the CMD. This process involves verifying the device's identity before retrieving its Protection Profile from the manufacturer's public MUD Server.

By incorporating device-specific policies, this system enhances the authentication process, allowing for real-time adjustments to network access based on policy changes. Consequently, it ensures a more adaptable and secure network environment for CMDs within the ENTRUST ecosystem.

4.2.2 ENTRUT Key Management

In this section we delve on key hierarchies. Each cryptographic key can be created as a primary key retrieved from a secret seed. Keys in a hierarchy have a parent-child relationship. A root key that is protecting other objects is called a storage parent key and the objects that it is protecting are its children. The ancestors of a key are the parent keys that connect the key to a primary seed. Keys are special-purpose keys such as storage keys, binding keys, sealing keys, Attestation identity keys (AIK), and signing keys as shown in Figure 4.1.

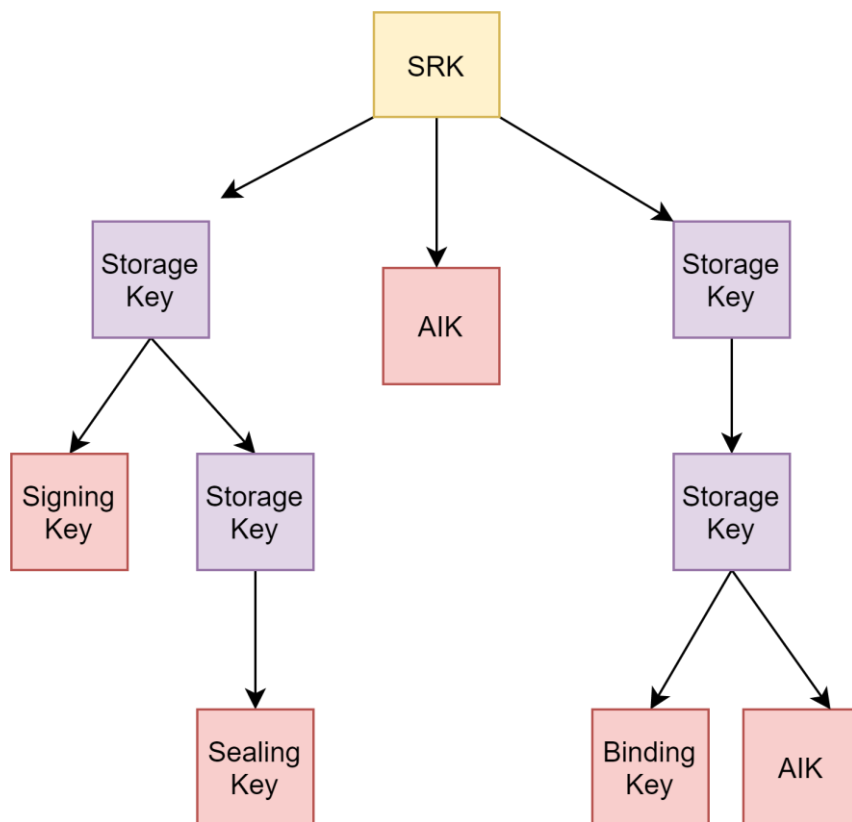


Figure 4.1: Key Hierarchy

The following is a list of the types of keys used in the ENTRUST framework. It details the functionality of the keys, whether the key is stored in the key hierarchy, usage/interactions with components of the architecture, the challenges in design relevant to ENTRUST, and potential security vulnerabilities if

compromised.

- **Root ID Key:** This is a symmetric key that is hardcoded in CMDs hardware during manufacturing. The Root ID Key, alongside with some other possible identifiers, is being used for the authentication of the device.
- **Identity Key (IK):** This key is derived from the Root ID key in combination with other unique properties of the device, aiming at more efficient authentication of CMDs. This is a symmetric key generated from an RNG during device manufacturing. The IK is an output of a Key Derivation Function KDF that takes RNG and other properties such as the hash of the secure boot as an input. Since the manufacturer knows the challenge-response pairs of the PUF and the output of the secure boot is part of the eMUD, we can derive a key that is able to authenticate a device in a manner not based solely on identity but to other properties as well. This key allows us to bootstrap trust on the medical device. It should be noted that this holds for the low-end CMDs that their TCB consists of a PUF. In the case of high-end CMDs, the RNG of the PUF is being substituted with the hardware device key of the TEE or its variants while the other parts of this procedure remain the same.
- **Hardware-Based TCB Keys:** These are the hardware-based master keys of TEEs. Their primary use is for creating a sealing interface for enclave storage of authenticated encrypted data such that keys depend on the platform and identity of the enclave.
- **SW/FW Update (Symmetric) Key:** This key is used to verify the integrity of SW/FW, ensuring both confidentiality and authentication. Each SW/FW service provider possesses such a key. During the update process, there are two possible types of signatures. In the first case, the SW/FW update provider signs the update with their private key, and the CMD verifies the correctness of the signature with its public key. This represents a one-to-many update process. On the other hand, there can be a pre-established authenticated encryption key for one-to-one updates.
- **Attestation Keys:** In the key hierarchy a key derivation function is required to derive the attestation identity keys from the root identity key of the platform. These keys are also signing keys used to provide a signature on an attestation message or assertion that ensures the trust level of a device. Those keys are binded to key restriction usage policies originating from the MUD profiles.
- **Attribute keys:** Every attribute is associated with a key, specifically, a signing key that is certified by a trusted authority in the device enrolment phase that will be presented in Story-V. These attribute keys will be used in the Attribute Based Access Control (ABAC) mechanism that includes some cryptographic schemes with attribute keys such as Attribute-based Encryption ABE, Attribute-based Signature ABS, or Attribute-based Signcryption which is a combination of ABS and ABE schemes in one protocol,
- **Ephemeral and Session Keys:** A single-use symmetric key is used for the encryption of all messages in one communication session. In ENTRUST, devices may be resource-constrained, meaning the use of public-key cryptography may not be possible. Session keys are useful in this context, as they improve the overall performance of encryption and limit the amount of information encrypted under one key. This is beneficial to security in case of key compromise. They can be used for establishing secure and authenticated channels within a service graph chain or between different chains. Such keys are ephemeral if they are generated for each execution of a key establishment process. They can be used more than once per communication session.

4.2.3 Crypto primitives and protocols for continuous secure and authenticated communication

Privacy-preserving Attribute-Based Credentials (p-ABC), also known as anonymous credentials, allow the achievement of privacy goals like minimal disclosure and unlinkability while keeping formal guarantees of the validity of claims over attributes. They were first envisioned by Chaum [20, 21]. There exist multiple instantiations, many of them based on Camenisch-Lysyanskaya (CL) [14, 17] or Pointcheval-Sanders (PS) [55] signatures. They provide various features like distributed issuance [12, 60], inspection [57, 15] or pseudonyms [13].

The W3C's Verifiable Credentials specification (VC) [69] is gaining significant traction in the current trend towards decentralization. This specification aims to establish a machine-verifiable, secure, and adaptable method for representing digital credentials. Its appeal lies in the interoperability and extensibility it offers, leading to widespread adoption across various domains such as data spaces [1] and the Internet of Things (IoT) [29]. However, the direct application of this specification can pose notable privacy concerns [36, 11, 32]. Integrating privacy-preserving Attribute-Based Credentials (p-ABCs) with VCs offers a twofold advantage: it simplifies the integration of p-ABCs, addressing one of their primary drawbacks, while also mitigating the privacy issues inherent in VCs as a digital credential specification.

Attribute-Based Credentials have seen few applications in IoT scenarios, mainly because of poor efficiency and lack of standardization. For instance, Canovas *et al.* [65, 66] proposes the use of Idemix [16] in IoT devices for attribute-based authentication. However, the lack of efficiency of the solution makes it impractical in most scenarios. A similar issue appears in [7]. Additionally, it fails to adapt to the dynamicity of IoT environments. Moreover, Lin *et al.* [44] found an attack on the proposed system. In [33], the authors apply a more efficient p-ABC solution in IoT scenarios, and signify the importance of its integration with Verifiable Credentials [69] and flexible configuration through device profiles for its effective adoption. The emergence of this standard, along with the recent interest on standardization in topics related to p-ABC¹ signifies a more fertile landscape for the adoption of p-ABCs, particularly on IoT scenarios. Nonetheless, the application of p-ABCs to improve privacy and security in IoT scenarios is still lacking, especially in topics outside the typical identity management such as trust management, conformity to certification and attestation.

After securely enrolled in ENTRUST services, every CMD is granted a VC that certifies a set of attributes, such attributes are needed to access safety-critical applications and information. To regulate the access control processes to some sensitive medical data, ENTRUST will develop cryptographic constructions that allow the system to verify that a CMD possesses specific attributes needed to access safety-critical applications and information even if committing his attributes. For example, a patient is only able to access or upload some sensitive data on the blockchain ledger if it has some identified attributes. This approach not only prevents unauthorized entities from accessing sensitive data but also offers this in a privacy-preserving manner where the CMDs can verify that they hold the required attributes without revealing them.

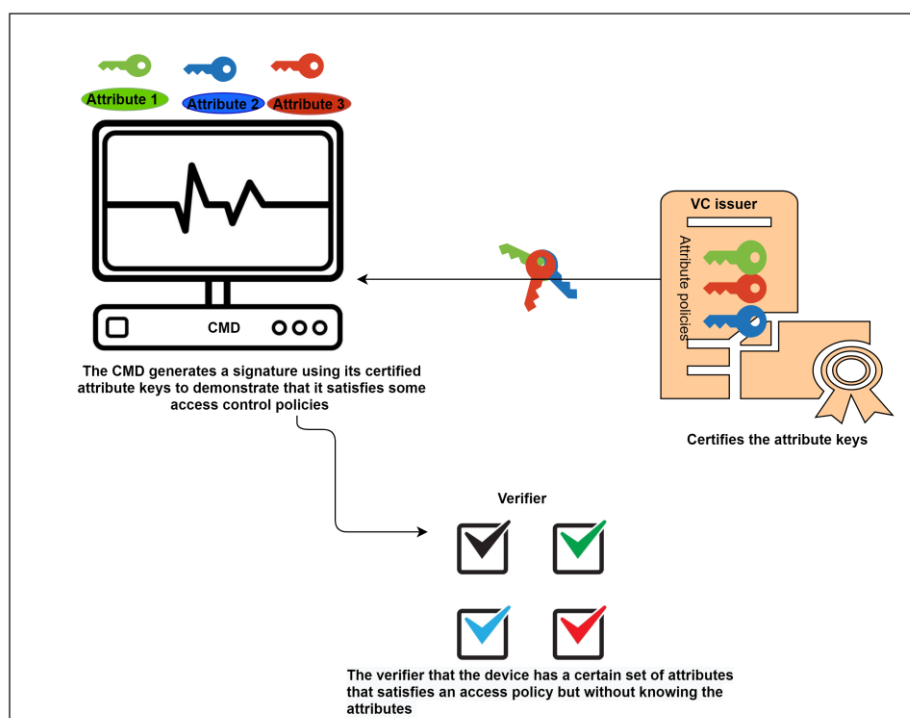


Figure 4.2: Attribute Based Access Control (ABAC)

In general, a CMD must be able to convince a verifier that it possesses some attributes even without revealing its identity or the attributes as described in Figure 4.2. This crucial ENTRUST requirement can be offered by attribute-based crypto protocols such as Attribute-based Signatures (ABS), Attribute-based Encryption, or even Attribute-based signcryption (ABSC) protocol that combines both ABE and ABS in one algorithm such as in [45].

Attribute-based signatures (ABS) introduced in [48], verifiers are convinced that the signer owns a set of attributes satisfying a so-called signing policy, however, they do not learn the signer's identity, nor the set of attributes used. ABS provides the signer's anonymity within a set of CMDs holding policy-conforming attributes. In ABS, CMDs cannot forge signatures with attributes they do not possess even through colluding.

To ensure confidentiality of the medical data stored on ENTRUST blockchain services, data is encrypted before being stored. ENTRUST will follow a decentralized approach that allows CMDs to decrypt the stored data based on their pre-certified attributes. This approach avoids introducing a central decryption authority that checks each CMD's attributes and then sends the decrypted data to the CMD if his attributes match some predefined decryption policy. To ensure decentralized access control to the stored medical data, ENTRUST will adopt Attribute-based Encryption (ABE) where a CMD has decryption rights to the encrypted data stored on the ledger only if he possesses some attributes that make correct decryption as introduced in [9]. This is an efficient construction that allows owners to share data safely with the designated CMDs rather than a third party or other entities.

¹E.g., <https://isrsm.gov.mk/en/project/show/iso:proj:88127>, <https://www.iso.org/standard/72018.html> or <https://www.iso.org/standard/80615.html>.

4.3 Engineering stories

Story-I: As a DM, I want to be able to have access to a key management system that can support the secure construction, maintenance, management, update and destruction of all the types of keys needed for a CMD to secure its lifecycle.

Objective: The main objective is to allow different ENTRUST parties to attest to the presence of a valid Root of Trust (RoT) for achieving the provision of root keys and certificates at the time of registration and during operation. The root keys are used to authenticate the device and then create the Attestation Keys and credentials bound to a policy depicting the correct configuration state that the device needs to be in before being allowed to use these restrictive signing keys. Essentially, this type of keys are the ones that were described beforehand and they need to constitute the key hierarchy of the device for allowing the device to provide evidence on its trust level without disclosing any information. This is achieved through the key restriction usage policies (see Story-X). Also, it should be specified whether there can be a link between some of the keys that belong to the same CMD or group of CMDs so that resulting signatures can be linked back to their origin. This is crucial for later revoking the keys of a device when needed.

Motivation: Key management is the process of managing cryptographic keys within a cryptosystem. **It deals with generating, exchanging, storing, protecting, and replacing keys.** Successful key management is critical to the security of a cryptosystem with the increased dependence on cryptography in everyday applications. It is very important and challenging to keep cryptographic keys safe and secure. A compromised single key could lead to a massive data leak with consequential reputational damage and loss of CMD confidence. Thus, reliable key management should establish and specify rules to protect the source's confidentiality, integrity, availability, and authentication. In the context of ENTRUST, the motivation is to be able to leverage the identified TCB for securing the management of the keys from the moment they are created to their decommission or destruction. For instance, a device that is equipped with a Root of Trust like a TEE, its keys must be managed by its security monitor and they need to be exposed only to those authenticated enclaves and processes. On the other hand, if a device is equipped with a PUF, there needs to be a scheme that enables the creation of the keys in such a way that can be verified by any outsider that the key belongs to the correct entity.

Building Blocks: ENTRUST framework may require different types of keys to be stored in the device such as Attestation Keys, encryption keys, and Blockchain keys. For example, a Trusted Component is identified by its Hardware-based TCB Key, while it uses an Attestation Key to provide attestation services such as signing a set of platform configuration registers and providing a timestamp. A Blockchain key is used to access ENTRUST Blockchain services and protect the communications between all the edge devices with the backend infrastructure. ENTRUST will support all necessary mechanisms for managing the key functionalities and indicate whether there can be a link between some of the keys (derive keys from each other using the trusted component's key hierarchy to reduce the key overhead).

In ENTRUST, as discussed, a key management system will be implemented. This system will be part of the secure monitor (trusted world) of a TCB. ENTRUST will provide a set of harmonized interfaces following the GlobalPlatform set of APIs that regard key creation and management. The core functionalities needed from a key management system, such as key creation, usage, binding, restricted usage, revocation, or authenticated usage, will be provided by using or extending the already defined interfaces by GlobalPlatform. Essentially, ENTRUST will create a key management system tailored to the resource constrained devices such as CMDs. This involves the creation of keys based on templates and the connection of keys as part of a key hierarchy.

Story-II: As a SP I want to have the necessary guarantees of a device's trust properties of interest when authenticated and registered into an operational domain.

Objective: The objective of the engineering story is the creation of the necessary mechanisms that guarantee the correct state of a device, as characterized by its manufacturer when it is authenticated and registered into an operational domain. It includes the design of robust policies that enable the continuous monitoring of the device, in order to prove that it runs in a way that protects the sensitive nature of medical data. The main functionalities that make this possible is the formally verified trust models and the Protection Profiles that are provided by the ENTRUST platform.

Motivation: The motivation behind ensuring the trust properties of interest of the device is that when it is authenticated and registered into an operational domain, there are guarantees that it remains uncompromised. If equipped with the necessary cryptographic primitives, we can assess its actual trust level at any point in time. The design of formally verified trust models that include cryptographic primitives and operations leads the CMD to meet the necessary security, privacy and trust requirements. Those models provide the ENTRUST platform with cryptographic and mathematical tools that detail the CMDs state. They include the assurance of the keys needed to be generated during bootstrapping (attestation keys) from the TC-based ENTRUST middleware. The end goal of those models is the establishment of clear trust boundaries for the operation of CMDs and the offering of verifiable guarantees on the correctness of their functioning. The other cornerstone of this engineering story is the creation of the Protection Profiles by the manufacturer and their polishment of the security administrator of the operational domain that the CMD is deployed. This engineering story is focused on the cryptographic aspects of those profiles and it is directly linked to [!! here mention the relevant D3.1 engineering story]. The core of the protection profiles is the specification of the various cryptographic primitives that should be attested during runtime verification. They include the necessary attestation challenges and sources of trust for each function to be attested. In summary, they define the specific operational parameters that need to be attested during runtime verification. This approach ensures the specification of the different sources of trust that define the correct state of the CMD and its uninterrupted usage in the operational domain.

Building blocks: During both the Design and Pre-deployment stage, the components that enable the functionality of this engineering story remain the same. First of all, the Formal Verification component designs the formally verified trust models of the CMD (see D3.1). Then, after the procedures performed by the Risk Assessment module (which are part of other engineering stories in D3.1), the Protection Profiles are composed. This is a task done by the Protection Profile Translator which

takes as input the RTL of the device and the formally verified trust models and creates eMUD files (more details can be found in D3.1). Afterwards, the profiles are sent to the MUD server. During the Design phase, a recalculation of the eMUD is necessary most of the times. The same steps are involved with the exception of the additional fetching of the existing eMUD from the MUD server and the communication of the Security Administrator with the DM. As a final step, the DM sends the Protection Profile to the Ledger. All the communication channels described need to be authenticated and protected with strong cryptographic guarantees. Those will be developed and described in D4.2.

Flow of actions: An overview of the expected flow can be found in Table 4.1 and Figure 4.3



Figure 4.3: Flow for Story-II: As a SP I want to have the necessary guarantees of a device's trust properties of interest when authenticated and registered into an operational domain.

	Trust Properties of CMD
Step	(1) Formal verification
Description	Design of the formally verified trust models for the CMD.
Origin	Formal Verification
Destination	Risk Assessment
Interfaces	-
Step	(2) RTL calculation
Description	Risk Assessment calculates RTL
Origin	Risk Assessment
Destination	Protection Profiles Translator
Interfaces	-
Step	(3) Formulation of eMUD
Description	Protection Profile Translator outputs eMUD.
Origin	Protection Profile Translator
Destination	MUD server
Interfaces	-
Step	(4) eMUD storing
Description	eMUD file is stored to the MUD server.
Origin	Protection Profile Translator
Destination	MUD server
Interfaces	-
Step	(5) eMUD fetching
Description	(If in Pre-deployment phase) DM fetches eMUD from MUD server
Origin	MUD server
Destination	DM
Interfaces	-
Step	(6) Domain-specific customization
Description	Security Administrator Customizes Profiles
Origin	DM
Destination	Security Administrator
Interfaces	-

Table 4.1: Flow for *Story-II: As a SP I want to have the necessary guarantees of a device's trust properties of interest when authenticated and registered into an operational domain.*

Story-III: As a security administrator, I want to be able to securely enroll the device to the domain manager in an autonomous manner

Objective: The objective of the engineering story is to ensure the secure and privacy-preserving boot-

strapping, authentication and registration of devices into the operational domain of ENTRUST. The focus is on creating a reliable and automated mechanism that can provide the necessary guarantees about the correct state of a device, as characterized by the manufacturer, before it is allowed to participate in the functional profile of the overall system. This will be accompanied by the establishment of the necessary cryptographic primitives, such as authentication mechanisms, to ensure that it can securely participate during runtime.

Motivation: The motivation behind securing privacy-preserving authentication, bootstrapping and the registration process is driven by the need to ensure that a device can be safely registered, and that there are guarantees that it remains uncompromised. The bootstrapping and registration process of the device must be secure to avoid the introduction of malicious or erroneous elements into the operational domain. To do so, it is necessary to establish a way to identify the device, that is, verify the device's root identity (e.g., asymmetric certificates installed by the manufacturer or crypto-primitives like PUFs) with the support of the manufacturer domain. The initial authentication will be based on an enhanced version of the EAP-AAA, details on the exact scheme and alternatives will be provided in D4.2. Additionally, the bootstrapping and registration must result on the creation of identity material for its use in the operational domain, such as additional key material e.g., identity/attestation keys. In this sense, following successful authentication and key creation, a pivotal enhancement involves engaging the Privacy CA to issue verifiable credentials, which will enable privacy-preserving attribute-based authentication, enhanced through ZK proofs. As part of the pursuit of strong security measures, several enhancements are underway in our system. The EAP-AAA-like authentication during bootstrapping is utilized to secure the extraction of policies from the MUD profile through the MUD manager (more information about MUD in Story-XVI). These policies serve as the foundation for creating key restriction policies, crucial for safeguarding device integrity (Story-X). The created material is zero-knowledge enabled. That is, it supports ZK proofs so that during the runtime phase operations can occur with complete guarantees but safeguarding privacy.

Building blocks: The device initiates the onboarding process by requesting inclusion into the target domain. The domain manager responds with an Authenticate request, with various alternative ways of covering the authentication process (asymmetric keys and certificates, PUF, pre-shared-keys with manufacturer). Upon successful authentication, the assertion is sent to the Privacy CA for enrolment and the provision of appropriate crypto primitives, namely Verifiable Credentials that establish identity attributes. Simultaneously, the device's profile and policies are requested from the MUD server, validated and applied. Verification against the ENTRUST blockchain determines if the device is in the correct state; failure results in onboarding termination. If successful, the Privacy CA executes an attestation JOIN, establishing an attestation key binding according to key restriction usage policies. A final check ensures the correct setup of these policies. Failure leads to authentication and registration failure, while success completes the authentication and registration process.

Flow of Actions: An overview of the expected flow can be found in Table 4.2 and Figure 4.4.

Enrolment	
Step	(1) Device initiates enrolment
Description	Device requests to be authenticated and registered into the target domain.
Origin	Device

Destination	Domain Manager
Interfaces	-
Step	(2) Domain manager authentication request
Description	The domain manager replies to the enrolment request with an authentication request
Origin	Domain manager
Destination	Device
Interfaces	-
Step	(3) Device is authenticated
Description	Device performs the authentication. Various alternatives may occur depending on previous steps and capacities: PUF, various EAP-like variants. It may involve external entities for the full authentication process (e.g. at manufacturer side)
Origin	Device
Destination	Domain manager
Interfaces	-
Step	(4) Device profile data obtained through MUD
Description	The device profile, security policies... are retrieved and installed in the domain. The process requires various steps which are detailed in Story-XVI.
Origin	Domain Manager
Destination	MUD Manager
Interfaces	-
Step	(5) Device obtains cryptographic material
Description	After successful authentication the data is sent to the Privacy CA to obtain cryptographic material for the operational phase.
Origin	Device
Destination	Privacy CA
Interfaces	-
Step	(6) Device state is checked
Description	Previous to finalizing enrolment, the device state is checked according to expected values that were stored in the blockchain.
Origin	Privacy CA
Destination	Blockchain
Interfaces	-
Step	(7) Setting up key restriction policies
Description	The final cryptographic material for the device is generated and key restriction policies are set according to device configuration obtained in step 4.
Origin	Privacy CA
Destination	Device
Interfaces	-
Step	(8) Final check
Description	A final check for the proper setup of key restriction policies and generated materials is performed, after which the enrolment is deemed successful.

Origin	Device
Destination	Privacy CA
Interfaces	-

Table 4.2: Flow for *Story-III: As a security administrator, I want to be able to securely enroll the device to the domain manager in an autonomous manner.*

Story-IV: As a user I want to have the feature of data sovereignty resolving of full control of who can access my medical records.

Objective: The objective of this story is to identify the mechanisms towards safeguarding the medical records that are stored in the ENTRUST DLT. ENTRUST envisions an access control mechanism that should be employed to control which entities are able to access these sensitive data. In this way, the data sovereignty in the ENTRUST DLT will be realised, as there will be full control of the entities that may access the medical records. Specific access policies should be satisfied by the entities towards accessing the ENTRUST DLT.

Motivation: The motivation for using ABAC mechanism in ENTRUST DLT is to ensure that unauthorized access of entities in the DLT and the stored data will be prohibited. ABAC offers a granular approach to access control, allowing fine-tuning of permissions based on various attributes. By integrating with trusted external sources, ABAC ensures the validity of attributes, enhancing the integrity of access decisions. ABAC enables the implementation of dedicated access policies that rule the conditions under which entities can access medical records, ensuring that only authorized personnel with relevant attributes can access this information. Through the employment of such access policies, ABAC enhances the security and privacy of medical records. More specifically, in ENTRUST, an entity must have its attributes verified from ABAC service. Only if these attributes are verified, the Blockchain Certificate Authority (CA) will issue a new certificate to the entity that will be also verified by the Security Context Broker and the ABAC for entering the DLT.

Building blocks: When an entity wants to enter to the ENTRUST DLT for the first time, it is necessary to send its attributes, that have been issued as Verifiable Credentials, to the ABAC service that will verify the validity of the issued attributes. If the attributes are correct, the Blockchain CA receives a request to register the entity in order for this to receive a correct certificate. If the attributes have been issued by a valid issuer, the Blockchain CA issues a new certificate that is stored to the entity. When the certificate is successfully verified by the Security Context Broker and the ABAC, the entity can enter the ENTRUST DLT and access information in the Private Ledger.

Flow of Actions: An overview of the expected flow can be found in Table 4.3 and Figure 4.5.

Attribute-Based Access Control	
Step	(1) The entity initiates access request
Description	The entity sends its attributes to ABAC service.
Origin	Entity
Destination	ABAC service
Interfaces	-

Step	(2) Validation of attributes by ABAC
Description	The ABAC service checks the validity of the attributes.
Origin	ABAC service
Destination	Entity
Interfaces	-
Step	(3) Request for registration to Blockchain CA
Description	Since the attributes were valid in the previous step, the entity requests to register to Blockchain CA with its VCs.
Origin	Entity
Destination	Blockchain CA
Interfaces	-
Step	(4) Validation of the VCs of the entity
Description	The Blockchain CA checks the validity of the VCs of the entity.
Origin	Blockchain CA
Destination	Entity
Interfaces	-
Step	(5) Issuance of certificate
Description	Since the VCs in the previous step were valid, the Blockchain CA issues a certificate that is stored in the entity.
Origin	Blockchain CA
Destination	Entity
Interfaces	-
Step	(6) Request for accessing ENTRUST DLT to Security Context Broker
Description	The entity sends the certificate to the Security Context Broker in order to access the ENTRUST DLT.
Origin	Entity
Destination	Security ContextBroker
Interfaces	-
Step	(7) Check of the certificate
Description	The Security Context Broker checks the certificate of the entity with the support of ABACservice.
Origin	Security ContextBroker
Destination	Entity
Interfaces	-
Step	(8) The entity enters the ENTRUST DLT
Description	Since the certificate in the previous step was valid, the entity is granted access to the ENTRUST DLT.
Origin	Entity
Destination	ENTRUST DLT
Interfaces	-

Table 4.3: Flow for *Story-IV: As a user I want to have the feature of data sovereignty resolving of full control of who can access my medical records*

Story-V: As a Domain AAA server, I want the enrolled CMD to be able to establish its keys for secure communication within the domain

Objective: This Engineering Story is related to Story-III and concerns the ability of CMDs to establish, secure, and authenticated communication channels with the creation of the appropriate cryptographic keys, assuming that they are in a correct state (they meet the RTL). This means that during runtime, CMDs should be capable of establishing secure and authenticated channels either with other CMDs on the same domain or in other medical graph chains if and only if they can exchange information on their trustworthiness states. This translates to the creation of all the necessary cryptographic material (such as Attestation Key (AK) and attribute keys), as well as the issuance and management of Verifiable Credentials that include all the necessary policies and verified attributes for getting access to all of the offered services.

Motivation: In ENTRUST, we must have a key management system that enables the secure management of the entire lifecycle of all types of CMDs that are required in such a complex ecosystem. From the attestation and attribute keys that are established during onboarding of a CMD while bootstrapping its trust state to the runtime construction of symmetric keys for the establishment of secure and authenticated communication channels. This key management system must be protected within the TCB, relying on the underlying root of trust. We should be able to verify that the validity of the trusted component, in order to support such a system, and to be able to create and securely store all these keys as part of the ENTRUST key hierarchy.

Building Blocks: During runtime, based on the created trust policies, each CMD will run its attestation controls in order to create conformity certificates. Essentially, conformity certificates are verifiable credentials that contain information on the trustworthiness level of a CMD. So, after onboarding, it takes the verifiable credentials containing the attributes of the CMD and creates all the necessary cryptographic material. During runtime trust assessment, and using the necessary attestation controls, it performs local attestation and creates conformity certificates that expose (in a privacy preserving manner) its trustworthiness. When a CMD wants to communicate with another CMD, there are two variants: If this communication exists within the same service graph chain then a channel is established directly between the devices by exchanging conformity certificates and a key exchange protocol. If they belong in different service graph chains, they establish a symmetric key with their parenting node (DM) and delegate the establishment of the channel between the two DMs.

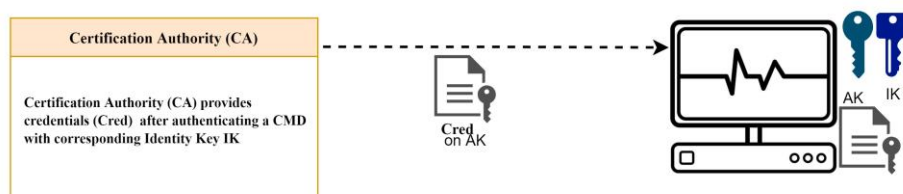


Figure 4.6: Secure Enrolment

Domain Enrolment	
Step	(1) Request Key Credential
Description	Each medical device should have a valid key credential issued by a trusted Certification Authority (CA). Such a credential allows the device to use its keys for attestation services.
Origin	Medical devices
Destination	CA
Interfaces	The device sends the public part of its attestation key together with its identity key to the CA to get a credential that allows the device to start using its attestation key.
Step	(2) Authentication
Description	The Certification Authority request the identity key certificate that was issued from the device manufacturer.
Origin	CA
Destination	Medical device manufacturer
Interfaces	The CA checks that the device has a valid identity key certificate
Step	(3) Credential Insurance
Description	The CA creates a signature on the public part of the device attestation key, using its signing keys. The signature constitutes the device key credential.
Origin	CA
Destination	Medical device
Interfaces	The medical device verifies the CA signature and stores the credential in its records.

Table 4.4: Domain Enrolment

Story-VI: As a high-end CMD, I want to be able to securely retrieve the Protection Profile of the CMD from the ledger.

Objective: The objective of this Engineering Story is to shed light to the distinction between low-end and high-end CMDs regarding the initiation of a runtime attestation procedure. This story is associated with Story-VII and concerns high-end CMDs. High-end CMDs, such as gateway devices, have the capability to perform asymmetric cryptography and are equipped with a secure element. They are more capable in terms of crypto operations than low-end devices and pose less overhead to the system when it comes to runtime attestation.

Motivation: As envisioned in ENTRUST, high-end medical devices, such as medical organization gateways, act as parent-devices that monitor and aid low-end devices in achieving the complex processes of the platform, from enrollment to attestation. They are capable of extensive

computational, attestation and cryptographic tasks. They are also equipped with the secure element to be used in ENTRUST and are capable of performing asymmetric cryptographic operations such as managing credentials and certificates, encryption, signatures etc.

Since the attestation challenges are described in the protection profile of a CMD, high-end devices must retrieve it from the ENTRUST ledger and extract any types of evidence they may need. This also happens to the case that the high-end CMD wants to perform operations supported by ENTRUST that regard "child" low-end devices (such as swarm attestation). Overall, it is envisioned that "parent" high-end machines will have "child" low end devices and will act as a proxy for the operations that the latter can't perform.

Building Blocks: In order for the high-end CMD to access the Ledger and retrieve its Protection Profile, it must be authenticated in the ENTRUST DLT. If this isn't the case, the device undergoes the process described in Story-IV. Afterwards, the CMD performs a Certificate-Based Authentication with the ledger, presenting the certificate issued by the Blockchain CA. If the Security Context Broker and the ABAC successfully verify the certificate, the device can enter the DLT and fetch the Protection Profile.

Flow of Actions: An overview of the expected flow can be found in Table 4.5 and Figure 4.7

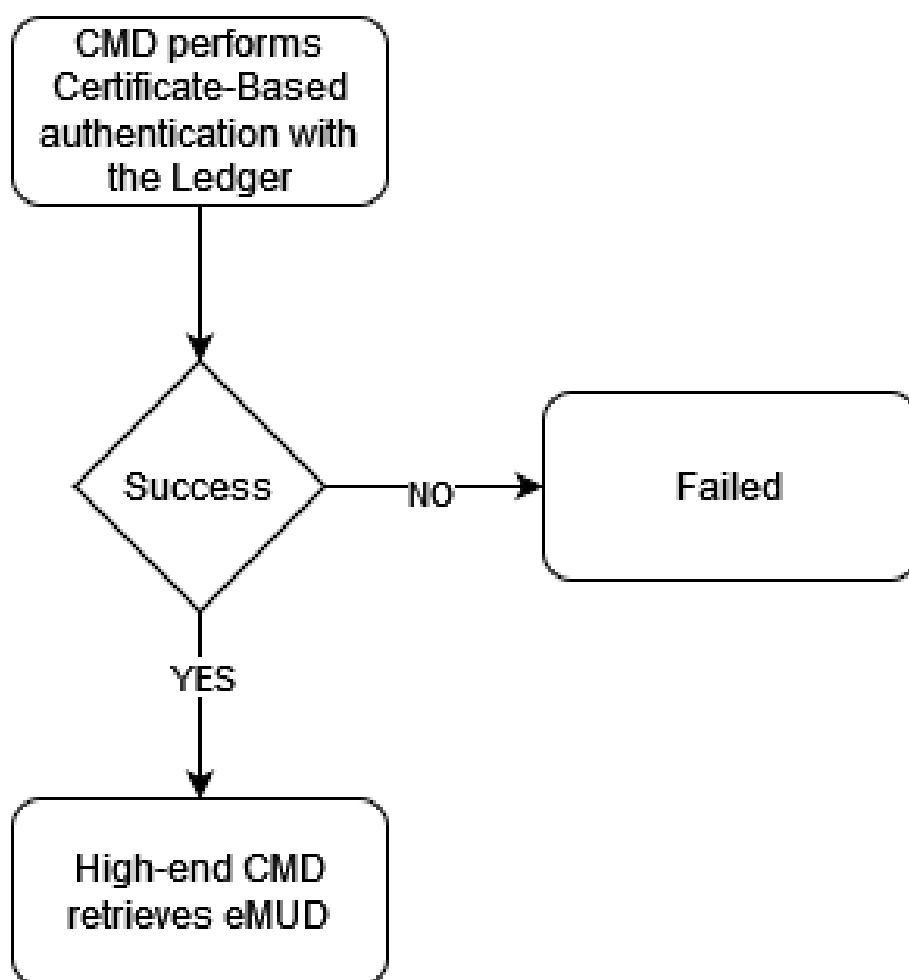


Figure 4.7: Flow for Story-VI: As a high-end CMD, I want to be able to securely retrieve the Protection Profile of the CMD from the ledger.

High-end CMD communication with Ledger	
Step	(1) Authentication
Description	CMD performs a Certificate-Based Authentication with the ledger, presenting the certificate issued by the Blockchain CA. If the Security Context Broker and the ABAC successfully verify the certificate, the device can enter the DLT.
Origin	CMD
Destination	Security Context Broker, ABAC
Interfaces	-
Step	(2) eMUD fetch
Description	The CMD fetches the eMUD in order to send it to the low-end CMD.
Origin	Ledger
Destination	CMD
Interfaces	-

Table 4.5: Flow for *Story-VI: As a high-end CMD, I want to be able to securely retrieve the Protection Profile of the CMD from the ledger.*

Story-VII: As a low-end CMD, I want to be able to securely communicate with the Domain Manager and collect the required types of evidence directly.

Objective: The objective of this Engineering Story is to shed light to the distinction between low-end and high-end CMDs regarding the initiation of a runtime attestation procedure. This story is associated with Story-VI and concerns high-end CMDs. Low-end CMDs, such as wearables, do not have the capability to perform asymmetric cryptography and are equipped with a PUF. They are less capable in terms of crypto operations than low-end devices and pose more overhead to the system when it comes to runtime attestation.

Motivation: The medical domains that the ENTRUST platform will support are expected to be large networks with multiple CMDs, all performing their assigned tasks in a secure, private and orchestrated way. Since in such complex environments, not all devices have the capacity to perform computationally expensive operations needed for security, ENTRUST differentiates low-end devices with minimal processing power from high-end devices that are described in Story-VI. In order to alleviate a part of this problem, we envision that each low-end device will be equipped with a PUF.

As described in the previous story, the low-end device may need to communicate with a high-end device that performs swarm attestation or other operations on their behalf. Also, as part of a remote attestation procedure performed by the device itself, it may need to communicate with the Domain Manager directly in order to collect the required types of evidence. Those methods of communication are performed with symmetric cryptography, since the low-end CMDs are equipped with a PUF that is only capable of producing symmetric keys. Overall, low-end devices' needs for attestation, and security in general, are performed with the help of the less demanding ways of

symmetric cryptography.

Building Blocks: The low-end CMD needs to be authenticated with the Domain Manager or the high-end CMD that acts on its behalf. When authenticated with the high-end CMD, the latter acts as highlighted in Story-VI, retrieving the Protection Profile from the DLT and providing it with the evidence needed for attestation. On the other hand, when the CMD asks the Domain Manager to provide it with the required types of evidence for attestation, it informs it about its MUD URL through the authenticated channel. Afterwards, the Domain Manager communicates with the DLT through its already established channel and retrieves the corresponding eMUD. Finally, it sends the attestation challenge and the corresponding sources of trust required to attest to the correctness of specific properties to the low-end CMD.

Flow of Actions: An overview of the expected flow can be found in Table 4.6 and Figure 4.8

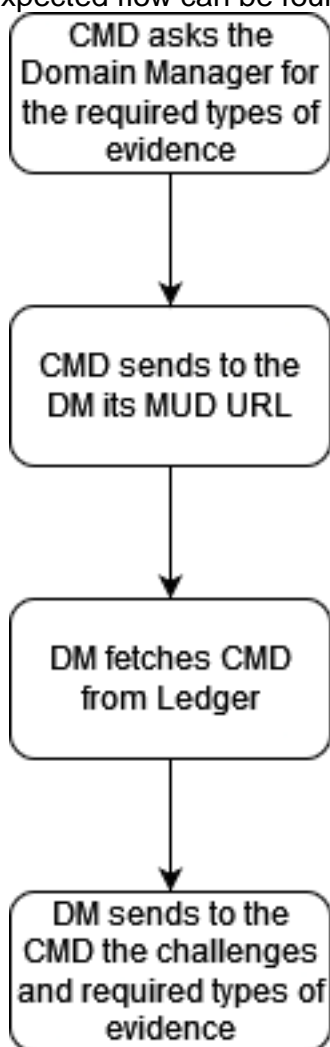


Figure 4.8: Flow for Story-VII: As a low-end CMD, I want to be able to securely communicate with the Domain Manager and collect the required types of evidence directly.

Low-end CMD communication with Ledger

Step	(1) CMD requests evidence
Description	CMD asks the Domain Manager for the required types of evidence
Origin	CMD
Destination	DM
Interfaces	Established authenticated channel
Step	(2) MUD URL info
Description	CMD sends the DM its MUD URL if necessary
Origin	CMD
Destination	DM
Interfaces	Established authenticated channel
Step	(3) eMUD fetching
Description	DM fetches CMD for Ledger
Origin	Ledger
Destination	DM
Interfaces	Established authenticated channel
Step	(4) Provision of challenges
Description	DM sends to the CMD the challenges and required types of evidence
Origin	DM
Destination	CMD
Interfaces	Established authenticated channel

Table 4.6: Flow for *Story-VII: As a low-end CMD, I want to be able to securely communicate with the Domain Manager and collect the required types of evidence directly.*

Story-VIII: As a manufacturer device provider component, I want to be able to audit and revoke Conformity Certificates in case of a change in the ATL in a privacy-preserving manner.

Objective: The objective of this engineering story is to describe the mechanism that allows revoking the attributes of a CMD in a privacy-preserving manner. In the case of device's ATL, ENTRUST will design a secure and privacy-preserving revocation protocol for de-activating some/all of the device's credentials, thus, excluding it from any further participation in some/all of the system activities.

Motivation: ENTRUST will focus on revoking the CMD's VC once a misbehavior or a change in the CMD attributes has been detected. For the correctness of the revocation scheme, we assume that the CMDs to be revoked were successfully enrolled with the Privacy CA and issued a valid VC that certifies their attributes. ENTRUST revocation deals with two revocation types:

- Revoking the CMD keys and credentials from the system in case of misbehavior.

- Partial revocation of some attributes of a CMD, for example revoking an attribute “Patient” after a user finishes a treatment at a specific hospital while keeping other attributes such as “age” and “address” unrevoked. This kind of revocation will be done in a privacy-preserving manner.

Building Blocks: In contrast with the traditional Public-key Infrastructures (PKI) that breach the privacy of shunned-out devices, ENTRUST will offer revocation of credentials in a privacy-preserving manner based on the creation of pseudonyms using the device’s attribute keys during the registration of each device. These pseudonyms are saved in the records and kept with some Revocation Authority (RA). If a Revocation Authority (RA) wants to revoke an attribute key from a device, it requests a signature using the device’s attribute key on a certain message. The RA then revokes the signature (with the corresponding key and credential) that links with a pseudonym found in the RA records.

Revocation	
Step	(1) Registration of pseudonyms
Description	Each medical device creates pseudonyms using its attribute keys.
Origin	Medical devices
Destination	The Revocation Authority (RA)
Interfaces	The pseudonyms are saved in the RA records.

Step	(2) Requesting a signature
Description	The device is asked from the RA to create a signature on a message using it attribute key.
Origin	Revocation Authority
Destination	Medical device with attribute key(s) to be revoked
Interfaces	The medical device create a signature using it attribute key.
Step	(3) Revoking an attribute key
Description	The device sends it requested signature to the RA
Origin	Medical device
Destination	RA
Interfaces	The RA verifies the signature then revokes the signature(with the corresponding key and credential) that links with a pseudonym found in the RA records.

Table 4.7: Revocation

Story-IX: Medical devices should be attested to verify that they have not been compromised. Each medical device should be able to provide verifiable evidence of its correct configuration state without disclosing the actual configuration

Objective: In the ENTRUST framework, provers should not be required to expose configuration and execution details as part of the measurements sent to the Verifier. Attestation and verification should be achieved in a privacy-preserving manner by enabling Provers to attest to their device configuration correctness and/or a critical program's correct execution in zero-knowledge while enabling the entire network to benefit from the security guarantees of the ENTRUST remotely verifiable attestation enablers.

Motivation: Enabling Verifiers to have full knowledge of a Prover's memory layout and configuration profile, such as the type of OS loaded, breaches user privacy and opens implementation disclosure attacks: "honest-but-curious" verifying entities can misuse sensitive information on the device's configuration, safety-critical functionalities, etc., which can be used to harm its general availability. Compounding this issue, what is needed, is the ability to verify the correct execution and/or configuration profile of a medical device without, however, being able to identify the actual device profile that is running.

Building Blocks: To achieve evidence privacy and authenticity in ENTRUST, we may investigate many approaches that include employing ring signatures as introduced in [61] that offer evidence authentication and evidence privacy by hiding the configuration of a device into a large set of configurations. Another possible approach to achieve evidence privacy is through employing Zero Knowledge Proofs such as SNARKs [49] where the prover proves that its device configuration matches the correct one without revealing its value.

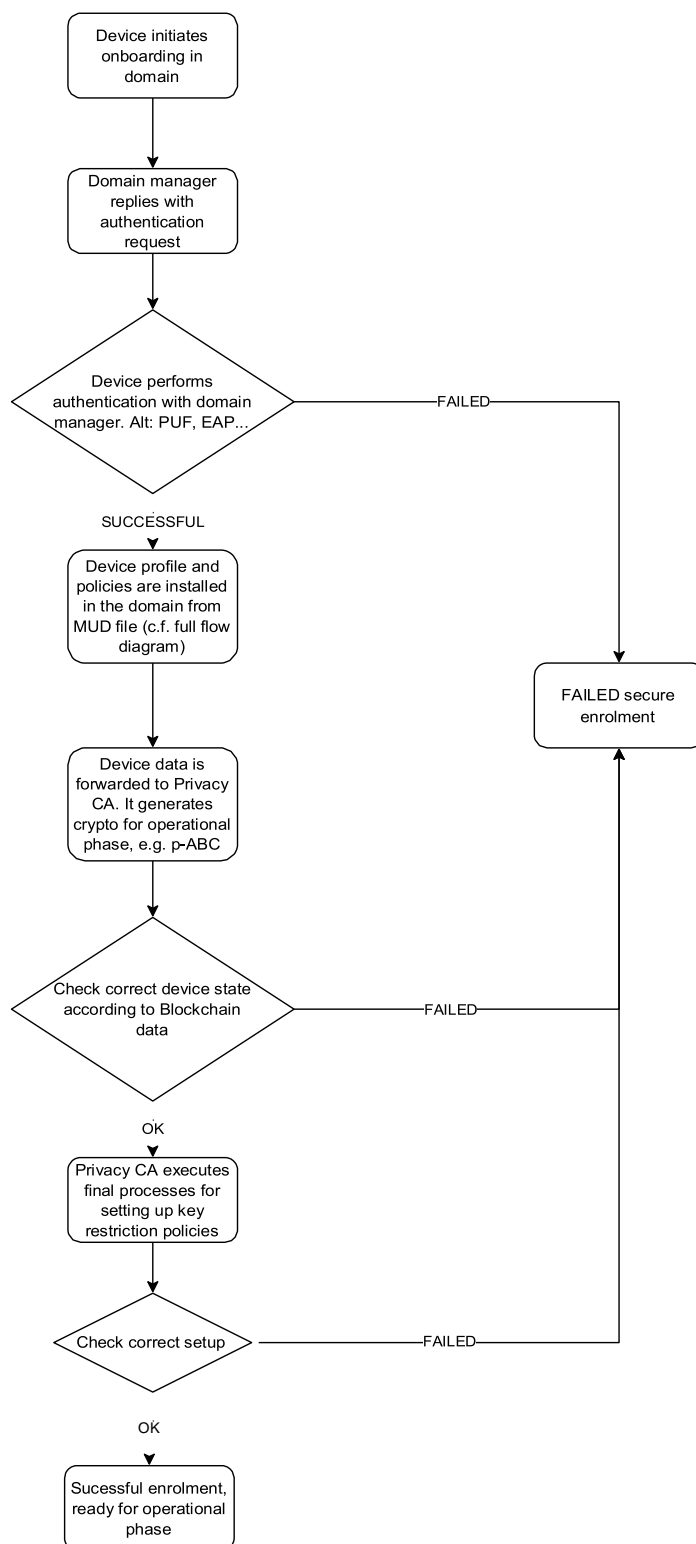


Figure 4.4: Flow for *Story-III: As a security administrator, I want to be able to securely enroll the device to the domain manager in an autonomous manner.*

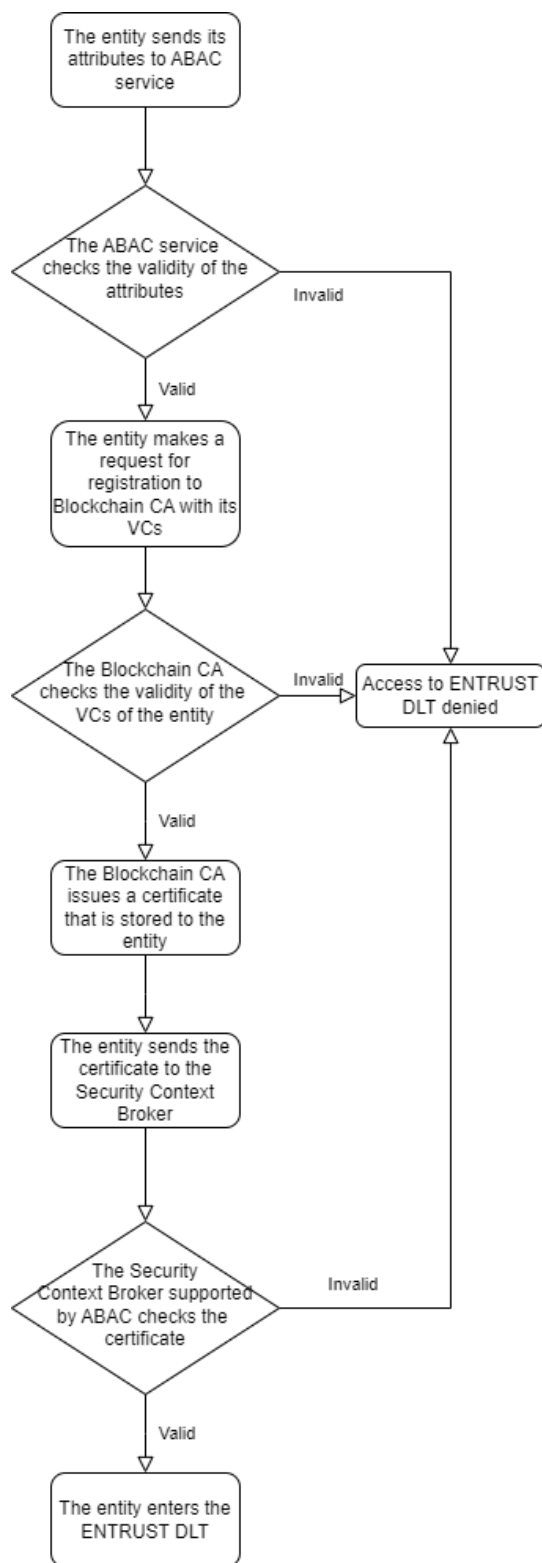


Figure 4.5: Flow for *Story-IV: As a user I want to have the feature of data sovereignty resolving of full control of who can access my medical records.*

Chapter 5

Trusted extensions

In continuation to the description of the crypto agility layer in ENTRUST, as presented in Chapter 4, this chapter provides the details of all those security controls needed for enabling a medical device to establish, during runtime, its level of trust. This is based on the underlying Trusted Computing Base, as defined in [Chapter 2](#), that is capable of providing evidence on the configuration and behavioral status of a CMD in a verifiable manner.

In ENTRUST, we provide new attestation mechanisms that enable the provision of various trust-related assurances on the configurational and operational correctness of a CMD in an efficient and privacy-preserving manner. This means that ENTRUST introduces the use of key restriction usage policies for enabling the assessment of the trust level in a zero knowledge manner. Essentially, the need for exchanging the raw related evidence, which can also lead to implementation disclosure attacks, is eliminated.

We provide attestation mechanisms as trust extensions that are efficient so that they can run on resource-constrained CMDs. As it pertains to attestation capabilities, the differentiating factor is how to be able to provide evidence in a verifiable manner. For a high-end CMD equipped with a rich TEE, capable of managing and coping with asymmetric crypto, attestation evidence can be signed through the appropriate crypto primitives described in [Chapter 2](#) and associated/linked to the device credentials. On the other hand, in the case of a low-end device which has capabilities for symmetric only crypto, the trust assessment needs to be mediated through a parent high-end node. This basically means that there needs to be a relationship established between a high and a low-end CMD, so that the former can forward the attestation evidence to the trust controls module on behalf of the latter.

5.1 State of the Art

5.1.1 Tracing techniques

Tracing techniques are crucial in various fields, including software development, cybersecurity, and performance analysis, thus it is an important aspect of ENTRUST. These techniques allow developers and analysts to monitor the behavior and execution of software applications, providing valuable insights into their operation. Among the prominent tracing techniques, taint analysis, binary instrumentation, and dynamic reconfiguration of binaries stand out for their effectiveness and versatility. Taint analysis is a dynamic program analysis technique used to track the flow of data through a program. It involves marking input data as "tainted" and tracing its propagation through the program's execution. Taint analysis can help identify potential security vulnerabilities, such as injection attacks or data leaks, by flagging tainted

data when it interacts with sensitive operations. One of the key benefits of taint analysis is its ability to provide precise data flow information, enabling developers to pinpoint the root cause of security issues.

However, taint analysis can be computationally expensive, especially for large and complex applications, and may generate false positives or miss subtle vulnerabilities. [25]

Binary instrumentation is another powerful tracing technique that involves injecting code into a binary executable to monitor its execution at runtime. By instrumenting the binary, analysts can collect detailed information about function calls, memory accesses, and control flow events. Binary instrumentation offers several advantages, including the ability to trace code paths that are difficult to observe using source-level instrumentation and the flexibility to analyze closed-source or third-party binaries. However, binary instrumentation can introduce overhead and perturb the target application's behavior, potentially leading to inaccurate results or performance degradation. [56]

Dynamic reconfiguration of binaries is a tracing technique that allows developers to modify the behavior of a running application without requiring source code changes or restarting the program. This technique is particularly useful for debugging and performance tuning, as it enables developers to dynamically enable or disable features, adjust parameters, or apply patches in real-time. Dynamic reconfiguration can enhance productivity by facilitating rapid experimentation and iteration during software development. However, it may introduce runtime overhead and complexity, especially when dealing with complex systems or interacting with external dependencies.

Tracing techniques such as taint analysis, binary instrumentation, and dynamic reconfiguration offer valuable capabilities for analyzing and understanding software behavior. By leveraging these techniques effectively, developers and analysts can gain deeper insights into the operation of software applications, identify potential vulnerabilities, and improve overall system reliability and performance. However, it is essential to carefully consider the trade-offs and limitations of each technique to ensure their successful application in practice. [42]

Transitioning from these foundational techniques, as applied in software development, cybersecurity, and performance analysis, the ENTRUST project seeks to pioneer advanced tracing methodologies tailored for the intricate ecosystem of connected medical devices. Central to this endeavor is the exploration of eBPF (Extended Berkeley Packet Filter) technology, coupled with custom Python scripting, to harness and analyze data flows within high-specification medical devices. This innovative approach promises to deliver granular, real-time insights into device operations, enhancing the detection and mitigation of potential vulnerabilities and performance bottlenecks. Moreover, ENTRUST is dedicated to addressing the paramount concern of integrity and authenticity within the tracing logs of lower-specification devices, which are ubiquitous in healthcare settings. The project intends to develop and implement robust mechanisms that not only ensure the reliability of tracing data but also safeguard it against tampering or unauthorized access. This dual focus strategy leveraging cutting edge tracing techniques for high-spec devices while ensuring data integrity for low-spec counterparts embodies ENTRUST's holistic commitment to fortifying the security and reliability of connected medical devices. Through these pioneering efforts, ENTRUST aims to set new benchmarks in the application of tracing technologies, ensuring that connected healthcare ecosystems are both resilient and trustworthy.

5.2 Engineering stories

Story-X: As a SP I want to be able to manage the key restriction usage policies in a verifiable manner.

Objective: This Engineering Story revolves around how ENTRUST allows the monitoring/sharing of trust related (attestation) evidence in a zero-knowledge manner. CMDs can provide attestation evidence protected through digital signatures from the attestation keys. The objective in this story is to be able to equip devices with the appropriate usage control policies on order to predicate the AK that is used to sign the various trust-related evidence. This is needed for sharing that evidence in a privacy-preserving manner. Essentially, this translates to a CMD performing local attestation and sharing its status in a verifiable manner by only disclosing a digital signature. This is because its AK is going to bind with an appropriate key restriction usage policy, representing its expected status. This story aims at providing a description of the trusted computing abstraction to establish policy restricted attestation keys that can dynamically restrict a CMD's AK secure in its TCB to policies chosen by an authorized entity. By predicating the CMD's ability for signing on its trust- worthiness information, we can verify the node's conformance using a simple challenge-response protocol that neither requires nor reveals its internal state. These solutions must be adaptable to immediate changes occurring during system onboarding and runtime, particularly concerning software or firmware updates.

Motivation: As already discussed, there is the basic need to update the policies that condition the running of a CMD, while preserving the ability to bind them to the key hierarchy. This has to be achieved without the recreation or alteration of existing keys. As an example, the key restriction usage policies that govern the AK should be able to be updated without altering the AK or the key hierarchy. This is a role that the Privacy CA should be able to play and as a result, to be in charge of updating the key restriction usage policies in a verifiable manner. This is especially crucial during a Software or Firmware update because it could result in the change of the expected state or trustworthiness of the CMD. Furthermore, it's imperative for the policies to incorporate extra security controls and versioning mechanisms to prevent attackers from exploiting outdated policies to create new potential attack vectors. Allowing such exploitation could lead to the generation of valid attestation assertions for invalid or outdated configurations, which are based on expired policies. In summary, within the context of ENTRUST, the necessity of establishing a resilient and verifiable method to ensure that policies are kept up to date alongside the CMDs' configuration and runtime attributes is evident. Therefore, it is equally crucial to enforce these policies securely and decisively.

Building blocks: The Attestation Agent and the Device Tracer are parts of the ENTRUST TCB. In the case of a high-end device, the Attestation Agent is running in its own enclave and is responsible for receiving the traces from the tracer during runtime. When onboarded in the target service graph chain, its hardware-based AK is generated and bind to the appropriate key restriction usage policy extracted from the eMUD. Those policies depict the expected state of the CMD which is provided by an authorized entity (Software Service Provider) as a reference value and is included inside the eMUD as a policy hash. Hence, during onboarding is extracted by the Domain Manager and the appropriate key restriction usage policy is enforced to the CMD and verified by the DM. Once this is achieved, then during runtime, the Attestation Agent is being configured with the respective trust policy dictating the type of attestation to be done and its periodicity. Hence the Attestation Agent triggers the Tracer for performing the runtime measurement and performs the local verification

against the established key usage control policies. We also need to consider the protection against one the pressing challenges against these types of solutions which is how to be able to restrict adversaries for keeping multiple key restriction usage policies active. This protocol gives the ability to update the key restriction usage policies in a verifiable manner without the need to update the keys of the CMD.

Story-XI: As a FW/SW Developer I want to be able to have the necessary guarantees to be able to ensure the secure update of a device.

Objective: The purpose of this engineering story is to capture one main concern of a possible FW/SW Developer that want to be certain about the security of their software or firmware. They need to be able to verify the trustworthiness of the code they are about to replace with their update. Additionally, the developer may need to know the actions to be taken in order to mitigate an error during the update or to revert the enclave to a previous state.

Motivation: In the context of ENTRUST, we need to guarantee the trustworthy state of the CMDs and their operations. To do so, different properties of the device are continuously attested. In case of a failed attestation, a new vulnerability may be discovered, leading to a mandatory update of the CMDs' software or firmware. Since the developer needs time to write their code and finish the update, the ENTRUST platform allows them to revoke to a previously stored copy of the enclave or keep the current vulnerable version of the CMDs' FW or SW.

As part of the previous description but also of the general SW lifecycle, the ENTRUST platform ensures that applications' version number will be included in their final package in the enclave. Nevertheless, this poses a possible security concern. Namely, a dedicated attacker may decide to reuse a previous application version that contains a known vulnerability. Thus, it is imperative to have a mechanism that prevents such attacks. The solution that ENTRUST offers is a sequential version number system that supports rollback protection while making sure that CMDs can't downgrade to vulnerable versions.

Building Blocks: The building blocks for this story will be examined in the next technical deliverables (D3.2, D4.2, D5.1) in which the ENTRUST Blockchain architecture and the corresponding SW/FW flow of actions will be described.

Story-XII: As a SP I want to be able to have assurances on the correct configurational and behavioral integrity of a device

Objective: The objective of this engineering story is the description of the actions that need to be taken from the ENTRUST platform for the continuous monitoring of the CMD. This is being done in order to ensure that the CMD is functioning correctly by the repeated checking against policies and information that are specified in the Protection Profiles. It is important to differentiate those components that are being checked from the ones that can be formally verified. The former can't provide well- defined policies during the device's onboarding. Consequently, in case of failure, the platform can take the required actions that are described in the other engineering stories.

Motivation: During the design and pre-deployment phase, multiple security-related operations of a CMD are formally verified and the appropriate sections in the Protection Profile are created. Nevertheless, this is not possible for all the components and operations performed of the CMD that is connected to the operational domain. Thus, it is crucial to have the appropriate attestation mechanisms that enable the continuous checking of the CMDs' state against predefined characteristics. This addition to the formal verification of the CMD provides a holistic approach to the protection of the privacy of CMD's attributes to prevent implementation disclosure attacks.

Building Blocks: During runtime, the Trust Controls component issues attestation challenges that monitor the CMDs' adherence to specified security requirements. The CMD executes the challenges that it is being sent to and as a result, confirms its' correct configuration and security state. The relevant evidence is collected from the Device Tracer as part of the attestation challenge. Next, the Attestation agent constructs a response to confirm the CMD state. Also, the Trusted Component plays a significant role, by the signing of the attestation report and the encryption of the collected traces. Additionally, the Wallet ensures the secure interactions of the CMD and the ledger. The key technologies here is the Attribute-Based encryption and Access Control that ensure the appropriate access levels to the information stored in the ledger, thus protecting the privacy of the patient.

Flow of actions: An overview of the expected flow can be found in Table 5.1 and Figure 5.1

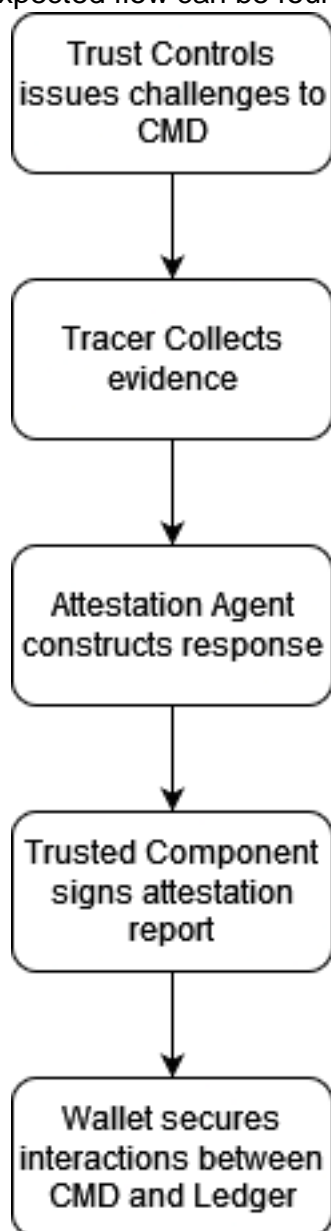


Figure 5.1: Flow for Story-XII: As a SP I want to be able to have assurances on the correct configurational and behavioral integrity of a device.

Configurational Integrity of CMD	
Step	(1) Issuance of challenges
Description	Trust Controls component issues attestation challenges that monitor the CMDs' adherence to specified security properties.
Origin	Trust Controls
Destination	CMD
Interfaces	-
Step	(2) Collection of evidence

Description	The relevant evidence are collected from the Device Tracer as part of the attestation challenge
Origin	Tracer
Destination	Attestation Agent
Interfaces	-
Step	(3) Response construction
Description	The Attestation Agent constructs a response to confirm the CMD state
Origin	Attestation Agent
Destination	Trusted Component
Interfaces	-
Step	(4) Attestation Report construction
Description	The Trusted Component signs the attestation report
Origin	Trusted Component
Destination	Trust Controls
Interfaces	-

Table 5.1: Flow for *Story-XII: As a SP I want to be able to have assurances on the correct configurational and behavioral integrity of a device.*

Story-XIII: As a Trust Controls module, I want to be able to audit the device status (during runtime in a verifiable manner)

Objective: The objective of this engineering story is to ensure that the Trust Controls module has the necessary mechanisms to audit the status of devices in the operational domain. It focuses on enabling the module to kick-start and ensure the attestation of the device status and will be complemented by the necessary cryptographic functionalities at the device's side.

Motivation: The Trust Controls module is in charge of the secure runtime operation of the domain. A key condition for this security is ensuring the safe state of devices operating in the domain. Thus, it is necessary for the module to be able to know the expected device status and perform the necessary processes to audit it. In particular, the specifics of the attestation process of a device will depend on its characteristics, so the manufacturer will need to support the domain by providing the necessary information for creating attestation challenges. Additionally, attestation of device status must be continuous, and it is important to ensure that the device status is assessed at the moment of attestation, and it is impossible for a device to forge the results. For this goal, it is first necessary that the attestation challenges are correctly updated, freshly generated for each process (e.g., avoiding reply attacks) and having enablers in the device established during the domain join that cryptographically protect the process.

Building Blocks: There are two main building blocks for the correct creation and usage of the attestation challenges. First, the attestation challenges are created according to key information about the device, which is obtained from the manufacturer through Manufacturer Usage Description (MUD) files. This MUD file will contain, along with protection profiles that describe security information about the device, information about conformance of the device to a desired trustworthy status that will be the seed for the audit process, including the creation of the challenges that will ensure a fresh audit process for the device. Second, the attestation challenges will be created according to the enablers in the device for secure attestation, e.g., through the use of key binding policies (c.f. reference attestation story) or according to the needs of the zero-knowledge proofs used by the device during the attestation process. For instance, the key restriction policies will ensure that challenges used during the audit process can only be replied to by the device if it is in the correct state, and challenges may include *nonces* for ensuring freshness.

Flow of Actions: An overview of the expected flow can be found in Table 5.2 and Figure 5.2.

Trust control audits device status	
Step	(1) Trust controls initiates audit process
Description	The Trust Controls module of the Domain Manager initiates the audit process for a device.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-
Step	(2) Check device profile and data
Description	The module checks the current profile and device data it has according to its validity period and data registered in the Blockchain.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-
Step	(2b) Obtain up-to-date profile
Description	If deemed necessary, the updated device profile, security policies... are retrieved and installed in the domain. The process requires various steps which are detailed in Story-XVI.
Origin	Domain Manager
Destination	MUD Manager
Interfaces	-
Step	(3) Attestation challenge
Description	The module creates attestation challenges according to the profile and device data and sends them to the device to initiate a new attestation process.
Origin	Domain Manager
Destination	Device

Interfaces	-
Step	(4) Device performs attestation
Description	The device responds to the attestation challenges with the results of the attestation of the device status.
Origin	Device
Destination	Domain Manager
Interfaces	-

Step	(5) Audit results
Description	The Trust Controls module audits the results of the attestation according to defined policies and expected trust level.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-
Step	(6a) Audit is deemed successful
Description	The audit process is successful as the policies and trust levels were met, finalising the audit process and leading to the creation and registration of the necessary proofs.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-
Step	(6b) Audit is deemed failed
Description	The audit of the device status is deemed failed because of issues in the device attestation or a failure by the device to meet the required policy. This will trigger the necessary response actions.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-

Table 5.2: Flow for *Story-XIII: As a Trust Controls module, I want to be able to audit the device status (during runtime in a verifiable manner)*.

Story-XIV: As a SP I want to be able to get access to runtime device system measurements in case of an indication of risk

Objective: The objective of this engineering story is to present the ways that the Service Provider has that enable them to evaluate the systems' state and the possible mitigation actions in case the

trustworthiness of the system is compromised. This story is linked with Story-XIII.

Motivation: As already mentioned in Story-XIII, the result of the runtime attestation is treated as a security claim, while the attestation evidence as the actual proof of the correct state of the CMD. When the Attestation Agent takes the traces that the Device Tracer provides, it forwards them, while taking into consideration the key restriction usage policies, to the Verifier. The Verifier sends the attestation report to the ledger to be recorded. After the completion of the verification, the CMD sends the raw tracer data to the Off-chain Storage Facility. The data are encrypted based on Attribute Based Access Control and the Domain Manager forwards the pointer of their location to the ledger. Based on the principles of Verifiable Presentations, anyone who has the appropriate set of Verifiable Credentials can create the necessary keys for the decryption of the attestation evidence. The Service Provider that has access to the raw traces can identify the execution path of the device. As a result, they can patch any entry points for the attack vector or identify zero-day vulnerabilities. It has become clear that human involvement stands as the final frontier, following design, when it comes to assessing the trustworthiness of a system and ensuring the CMD operates as intended.

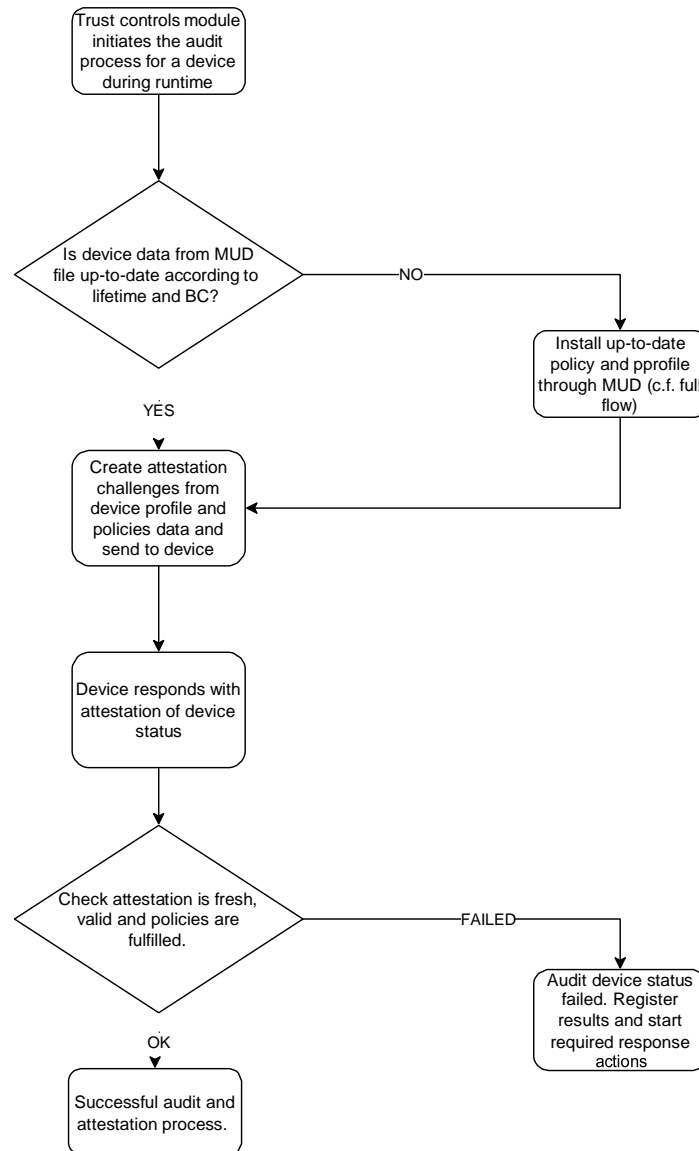


Figure 5.2: Flow for *Story-XIII: As a Trust Controls module, I want to be able to audit the device status (during runtime in a verifiable manner)*.

This thorough process guarantees that the attestation procedure is conducted systematically, with actions tailored to the device's status and potential risks. Prioritizing security throughout the system's design stages reduces the necessity for reactive responses to breaches of the trust model, as such occurrences are anticipated.

Building Blocks: The Certification and Auditing Component receives the attestation report, creates conformity certificates and utilizes Attribute-based encryption to secure the contents of the report. Then, the encrypted attestation report and the conformity certificate are transmitted to the DLT. The evidence is stored off-chain while a pointer that references its position is written in the ledger alongside with the conformity certificate. When needed, traces can be retrieved by dereferencing the aforementioned pointers. If an entity belonging to the ENTRUST platform can access the ledger, it can decrypt and evaluate the traces for further analysis and verification if it possesses the relevant attributes needed for decryption.

Flow of actions: An overview of the expected flow can be found in Table 5.3 and Figure 5.3

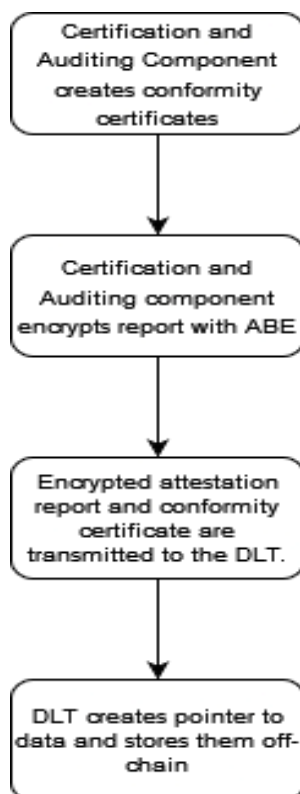


Figure 5.3: Flow for Story-XIV: As a SP I want to be able to get access to runtime device system measurements in case of an indication of risk

	Runtime device system measurements
Step	(1) Conformity Certificates Creation
Description	Certification and Auditing Component creates conformity certificates
Origin	Certification and Auditing
Destination	Certification and Auditing
Interfaces	-
Step	(2) ABE
Description	Certification and Auditing component encrypts report with ABE
Origin	Certification and Auditing
Destination	Certification and Auditing
Interfaces	-
Step	(3) Data transmission
Description	Encrypted attestation report and conformity certificate are transmitted to the DLT

Origin	Certification and Auditing
Destination	DLT
Interfaces	-
Step	(4) Data storing
Description	DLT creates pointer to the data and stores them off-chain
Origin	DLT
Destination	DLT and off-chain storage
Interfaces	-

Table 5.3: Flow for *Story-XIV: As a SP I want to be able to get access to runtime device system measurements in case of an indication of risk.*

Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.

Objective: The main objective of this story is to enable the creation of conformity certificates for devices so that the trust assessment that is continuously performed can be verified and registered.

Motivation: To provide a trustworthy and secure environment in the domain, the trust level of devices must be continuously assessed and monitored. For doing so, the device's status is audited through attestation methods (c.f., e.g., Story-XIII). The result of the audit of the verifiable attestation claims must be checked against information specific to the device's expected runtime characteristics. Thus, it will require a check for conformance to the protection profile and runtime conformity certificate information included in the device's profile defined through MUD files. If the check is correct, it is necessary to create a way for showing that the device is currently at an appropriate trust level. Thus, the emission of a Conformance Certificate is needed. This certificate must be publicly verifiable and contain information based on verifiable evidence on the security posture of the device.

Building Blocks: The building blocks are Verifiable Credentials for the creation of conformity certificates. They can be enhanced with zero-knowledge proofs (p-ABC) for improved privacy capabilities. Additionally, an indirect building block is the mechanisms necessary for generating and verifying the evidence of the device attestation for the trust level check.

Flow of Actions: An overview of the expected flow can be found in Table 5.5 and Figure 5.4.

	Trust control audits device status
Step	(1) Device audit process was deemed successful
Description	The audit process (c.f. Story-XIII) of the device is completed and deemed successful.
Origin	Domain Manager
Destination	Device

Interfaces	-
Step	(2) Generate report
Description	The Domain Manager generates and registers a report and verifiable evidence of the process.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-
Step	(3) Conformity Certificate
Description	A Conformity Certificate is generated according to the specifications in the MUD file, reflecting the conformity of the device to the security and trust level policies established for it. This generation is registered in the Blockchain.
Origin	Domain Manager
Destination	Blockchain
Interfaces	-
Step	(4) Device is notified
Description	The Conformity Certificate generated is forwarded to the device so that it can use it as verifiable proof of its trust status in the domain, thereby notifying the device of the success of the audit process.
Origin	Domain Manager
Destination	Device
Interfaces	-

Table 5.4: Flow for *Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.*

Story-XVI: As a domain manager I want the operational domain to have the capability to retrieve, adapt and apply security policies based on devices and threats.

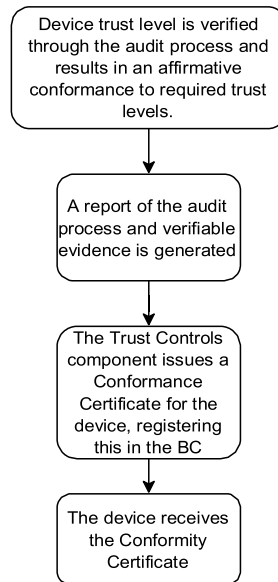


Figure 5.4: Flow for *Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.*

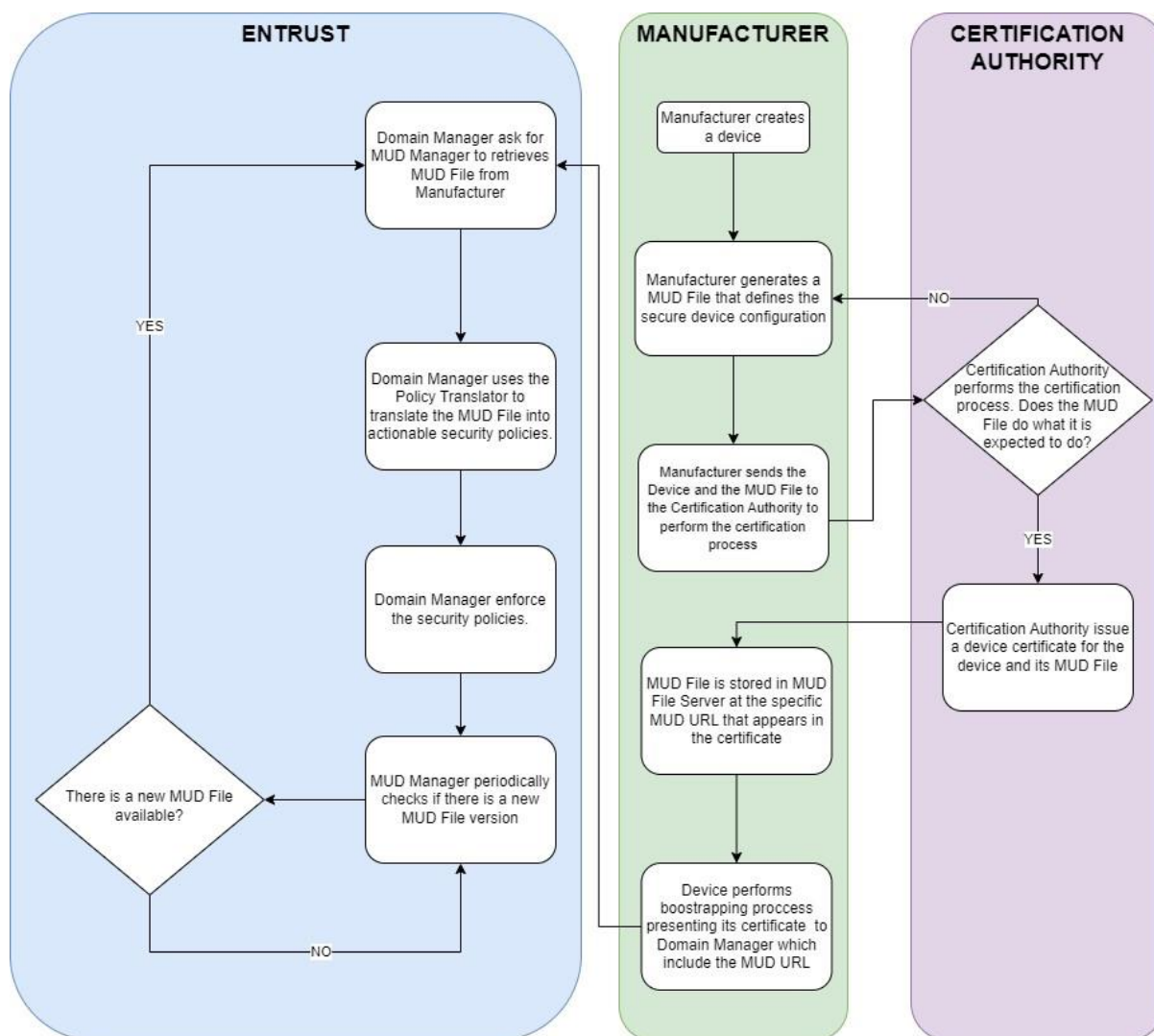
Objective: The primary objective of this engineering story is to enhance network security by developing a system capable of dynamically adapting and implementing security policies that are specifically designed for individual devices and emerging threats. This initiative aims to empower ENTRUST'S operational domain to effectively leverage the Manufacturer Usage Description (MUD) framework. The MUD framework outlines the intended behavior and network necessities of IoT devices, thereby enabling network administrators to make well-informed decisions about network traffic oversight and threat neutralization. The domain manager plays a crucial role in orchestrating this process, ensuring that security policies are not only retrieved and adapted but also efficiently applied to safeguard the domain's integrity.

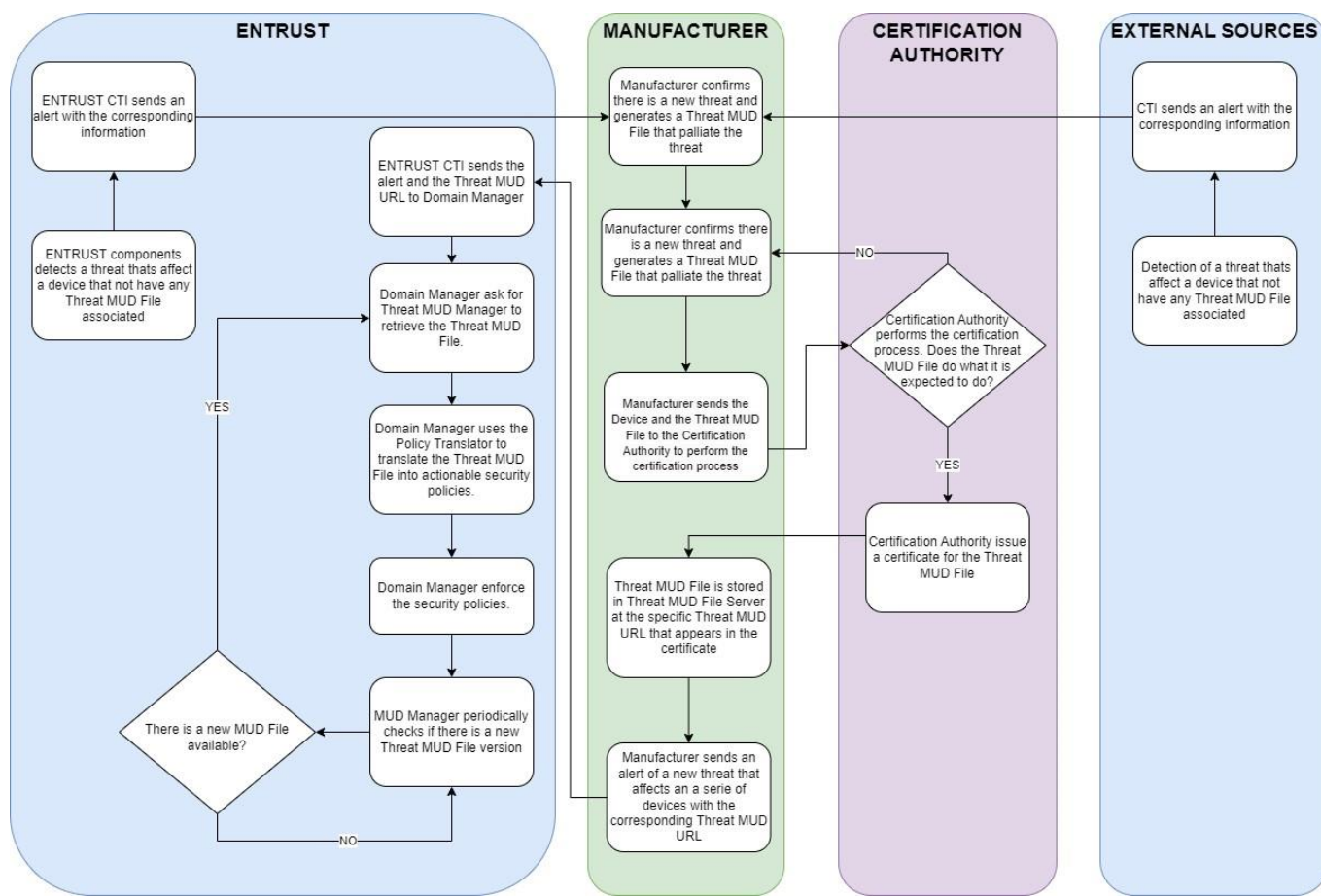
Motivation: The landscape of IoT and network security is rapidly evolving, presenting new challenges that demand robust and adaptive security solutions. Traditional security approaches often fail to address the unique challenges posed by the myriad of IoT device, each with its own set of vulnerabilities and operational behaviour. The ENTRUST project recognizes this gap and aims to bridge it by introducing a comprehensive security framework that leverages the Manufacturer Usage Description (MUD) and Threat MUD files. MUD represents a crucial development role in IoT security, offering a standardized method for device manufacturers to describe the intended network behaviors and communication profiles of their devices. By specifying the types of connections, a device should make, MUD allows network administrators to automatically apply rules that limit device communications to those that are necessary for its operation, reducing the attack surface and mitigating the risk of device exploitation. The introduction of Threat MUD Files extends this concept by allowing for the specification of behavior and communication patterns that are indicative of compromised devices or emerging threats. These files, informed by real-time threat intelligence, enable the Domain Manager and associated systems to adapt quickly to new vulnerabilities, ensuring that security measures evolve in tandem with the threat landscape. The ENTRUST project's utilization of both MUD and Threat MUD Files within its security framework marks a significant advancement

in IoT security. The MUD Manager and Threat MUD Manager are pivotal in this architecture, managing regular and threat-specific MUD Files, respectively. This dual-layered management system ensures that devices are not only configured with their intended operational profiles but are also continuously assessed and updated in response to new threats. The Domain Manager's role in overseeing the retrieval, adaptation, and application of security policies based on these files is critical for maintaining the operational domain's integrity and resilience against attacks. Through the meticulous integration of MUD and Threat MUD Files, the ENTRUST project offers a dynamic and proactive approach to IoT security. This strategy not only enhances the security posture of individual IoT devices but also strengthens the overall network's ability to withstand and respond to threats. By embracing this comprehensive solution, the ENTRUST project underscores its commitment to advancing IoT security, ensuring that devices across the network are better equipped to navigate the complexities of the modern threat landscape.

Building Blocks: We must differentiate among the MUD flow and the Threat MUD flow, in case of MUD: the process is initiated at the manufacturing stage, where a new device is manufactured. The manufacturer will generate a file where it will be specified the security profile with the appropriated configuration to be setup in a secure way. This File will be the MUD file which will be allocated in a common repository called MUD File Server controlled by the Manufacturer, on the other hand we have the device in which will be installed a certificate issued by a Certification Authority which will execute a certification process to certify that this MUD File its correct and secure. In this certificate will be included the MUD URL where its pointing to the MUD File at the MUD File Server. Once this device performs the bootstrapping, the device communicates its MUD URL to the Domain Manager, marking the start of the policy adaptation process. The Domain Manager acts as the orchestrator, processing this information and instructing the MUD Manager to retrieve the relevant MUD File. The Policy Translator's converts these files into actionable security policies that are then enforced within the domain. Overseen by the Domain Manager, this streamlined process ensures that the operational domain adeptly applies the most current and effective security measures to counter threats and maintain device integrity. In case of the Threat MUD Flow, when a new threat that affects the device is detected, the manufacturer will be notified via CTI, then, the manufacturer will generate a file where this threat will be palliated, this file will be the Threat MUD File, as the MUD File this file will be certified by a Certification Authority and once this file is certified it will be uploaded to the Threat MUD File Server pointed by a certain Threat MUD URL, this URL along with the Threat Identifier (CVE or similar) will be published via CTI. Once this information reach ENTRUST CTI consumer, this will be sent to the Domain Manager which will be sent to the Threat MUD Manager, this manager then will retrieve the Threat MUD File and then send this File to the Policy Translator to be converted in a actionable security policies that will be the enforcement.

Flow of Actions:





Domain Manager MUD and Threat MUD Operations

Step	(MUD 1) Manufacturer sends a new MUD File to Certification Authority
Description	When a new device is created or a new configuration is needed for an existing device, the manufacturer creates a new MUD File and send it to the Certification Authority t performs the certification process. After this step finish with a success certification, the MUD File and its signature will be uploaded into the MUD File Server at manufacturer's premises.
Origin	Manufacturer
Destination	Certification Authority
Interfaces	-
Step	(MUD 2) Sends MUD URL
Description	Device sends its MUD URL to the Domain Manager, this one store it and start the MUD process.
Origin	Device
Destination	Domain Manager
Interfaces	-
Step	(MUD 3) Domain Manager sends MUD URL to MUD Manager
Description	Domain Manager sends the device MUD URL to the MUD Manager.

Origin	Domain Manager
Destination	MUD Manager
Interfaces	-
Step	(MUD 4) MUD Manager retrieves MUD file and validates it
Description	MUD Manager goes to the MUD file server where MUD file is allocated and retrieves it. Then it performs a series of validations to check its certification process in order to trust that file.
Origin	MUD Manager
Destination	MUD File server
Interfaces	-
Step	(MUD 5) MUD Manager sends MUD File to Policy Translator
Description	MUD Manager sends MUD File to Policy Translator in order to convert it to actionable mitigation actions.
Origin	MUD Manager
Destination	Policy Translator
Interfaces	-
Step	(MUD 6) Policy Translator send the converted MUD File back to the Domain Manager
Description	Policy Translator translate the MUD File received into actionable mitigation actions and send it back to the Domain Manager.
Origin	Policy Translator
Destination	Domain Manager
Interfaces	-
Step	(MUD 7) Domain Manager enforce the mitigation actions
Description	Once the Domain Manager receive the mitigation actions from the MUD File, the Domain Manager enforce those mitigation actions in the ENTRUST platform.
Origin	MUD Manager
Destination	MUD File Server
Interfaces	-
Step	(Threat MUD 1) Manufacturer generates a Threat MUD File
Description	When Manufacturer detects or receives a new threat related to a certain device, its also generate a proper Threat MUD File to address this Threat, this file its send it to the Certification Authority to perform the certification process. Once all its certified, the Threat MUD File is allocated into the Threat MUD File Server at manufacturer premises and the Threat MUD URL its circulated via CTI.
Origin	Manufacturer
Destination	Certification Authority

Interfaces	-
Step	(Threat MUD 2) CTI sends there is a new Threat MUD URL available
Description	When CTI detects that there is a new Threat MUD URL that affects to any of ENTRUST devices, it automatically sent to Domain Manager.
Origin	CTI
Destination	Domain Manager
Interfaces	-
Step	(Threat MUD 3) Domain Manager sends Threat MUD URL to Threat MUD Manager
Description	Domain Manager sends the Threat MUD URL to the Threat MUD Manager.
Origin	Domain Manager
Destination	Threat MUD Manager
Interfaces	-
Step	(Threat MUD 4) Threat MUD Manager retrieve Threat MUD File and validates it
Description	Threat MUD Manager goes to the Threat MUD File Server where Threat MUD File is allocated and retrieve it, then it perform a series of validations to check its certification process in order to trust in that file.
Origin	Threat MUD Manager
Destination	Threat MUD File Server
Interfaces	-
Step	(Threat MUD 5) Threat MUD Manager sends Threat MUD File to Policy Translator
Description	Threat MUD Manager sends Threat MUD File to Policy Translator in order to convert it to actionable mitigation actions.
Origin	Threat MUD Manager
Destination	Policy Translator
Interfaces	-
Step	(Threat MUD 6) Policy Translator send the converted Threat MUD File back to the DomainManager
Description	Policy Translator translate the Threat MUD File received into actionable mitigation actions and send it back to the Domain Manager.
Origin	Policy Translator
Destination	Domain Manager
Interfaces	-
Step	(Threat MUD 7) Domain Manager enforce the mitigation actions

Description	Once the Domain Manager receive the mitigation actions from the Threat MUD File, the Domain Manager enforce those mitigation actions in the ENTRUST platform.
Origin	Domain Manager
Destination	Domain Manager
Interfaces	-

Table 5.5: Flow for *Story-XV: As a Trust Controls component, I want to be able to create Conformity Certificates validating the trust status of a device.*

Story-XVII: As a Gateway Tracer I want to protect the integrity and authenticity of the gathered data.

Objective: The objective of this initiative is to safeguard the integrity and authenticity of data gathered by Tracers within the operational domain of ENTRUST. The primary focus is on establishing a robust and resilient framework that guarantees the unaltered state and accurate representation of collected data from inception to analysis. This entails implementing stringent security measures to prevent unauthorized access, tampering, or corruption of critical information, thereby preserving the trustworthiness of our investigative findings. Additionally, the development of comprehensive cryptographic protocols will be paramount, ensuring that data remains protected and verifiable.

Motivation: The motivation behind ensuring the integrity and authenticity of the logs generated by a Tracer lies in the critical importance of maintaining a reliable record of events for attestation purposes. The reliability of the tracer findings is directly dependent on the trustworthiness of the logs produced within a specific execution environment (trusted/untrusted world). By prioritizing the integrity of these logs, it ensures that they accurately reflect the sequence of events, remain immune to tampering, and uphold their credibility as irrefutable evidence. Moreover, preserving the authenticity of the logs is essential for establishing transparency and accountability in our investigative processes. Attestation agents rely on the accuracy and trustworthiness of these logs to make informed decisions and pursue appropriate actions. Therefore, safeguarding the integrity and authenticity of the logs can reinforce the credibility of attestation process.

Building Blocks: The process begins with the device trust agent triggering the Tracer to commence collecting specific properties, initiating the tracing procedure. Once activated, the Tracer logs the desired results. To ensure the security and confidentiality of the gathered data, the Tracer encrypts the results using its designated keys before transmitting them to the attestation agent. Upon receipt, the attestation agent decrypts the Tracer log and proceeds with the attestation process to verify the integrity and authenticity of the data. The results of this attestation, along with the encrypted log traces, are then securely stored in the device wallet, safeguarding them for future reference and analysis. Through this meticulous process, each component fulfills its role in upholding the integrity and reliability of the data collected, ensuring the trustworthiness of the tracing as a whole.

Gateway Tracer operations

Step	(Step 0) Attestation process start
Description	The trust agent is informed by the domain agent to request the attestation process
Origin	Domain Manager
Destination	Trust agent Authority
Interfaces	-
Step	(Step 1) Trust agent request tracer to collect data and security claims
Description	The trust agent requests the tracer of a high-spec device to log runtime information and security claims
Destination	Tracer
Interfaces	-
Step	(Step 2) Tracer collects data
Description	The tracer collects runtime information and security claims
Origin	Tracer
Destination	Tracer
Interfaces	-
Step	(Step 3) Encrypt data with tracer key
Description	The tracer encrypts gathered data with its key.
Origin	Tracer
Destination	Tracer
Interfaces	-
Step	(Step 4) Tracer sends data to attestation agent
Description	The Tracer transmits the collected data to attestation agent.
Origin	Tracer
Destination	Attestation Agent
Interfaces	-

Table 5.6: Flow for Story-XVII: As a Gateway Tracer I want to protect the integrity and authenticity of the gathered data

Story-XVIII: As a Bare Metal Tracer I want to protect the integrity and authenticity of the gathered data.

Objective: The goal of this initiative is to protect the integrity and authenticity of data collected Tracer in low-spec devices within ENTRUST's operational field. The emphasis is on creating a strong and durable infrastructure that ensures the data's integrity from the moment it's collected to its final analysis. This will involve the application of rigorous security practices to block any unauthorized

entry, manipulation, or degradation of vital data, thus maintaining the credibility of its data they capture. Given the hardware limitations, instead of complex cryptographic protocols, the focus will shift to establishing a secure and verifiable firmware environment, ensuring the data's protection and integrity.

Motivation: The driving force for securing the integrity and authenticity of data from the Tracer of low-end devices is the paramount need for a dependable event record for verification purposes. The reliability of insights from these devices hinges on the integrity of the data they capture. Focusing on this data's integrity ensures it faithfully mirrors the event sequence, remains resistant to unauthorized changes, and maintains its standing as conclusive proof. Furthermore, the authenticity of this data is crucial for fostering clear and responsible attestation procedures. Decision-makers and verification agents depend on this data's precision and reliability to make well-informed choices and take necessary measures. Thus, protecting the data's integrity and authenticity bolsters the trustworthiness of the entire verification framework.

Building Blocks: The procedure initiates with the device's secure boot mechanism activating the Tracer which begins the data collection process. Once operational, the Tracer captures the necessary data. Given the hardware constraints of low-spec devices, instead of encrypting the data, the emphasis is on maintaining a secure operational environment through the integrity of the device's firmware. The collected data is then directly sent to the attestation agent without encryption. The attestation agent, relying on the secure initial state ensured by the secure boot, assesses the data to confirm its integrity and authenticity. This careful approach ensures that each step, adapted to the limitations of low-spec devices, contributes to the integrity and trustworthiness of the collected data.

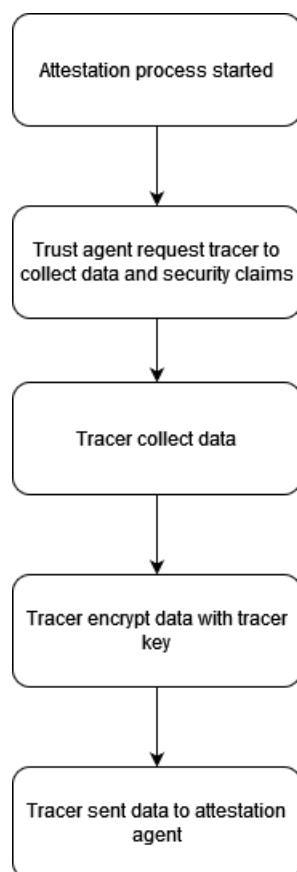


Figure 5.5: HS SPEC

Bare Metal Device Tracer operations	
Step	(Step 0) Attestation process start
Description	The trust agent is informed by the domain agent to request the attestation process
Origin	Domain Manager
Destination	Trust agent Authority
Interfaces	-
Step	(Step 1) Trust agent requests tracer to collect data and security claims
Description	The trust agent requests the tracer of a low-spec device to log runtime information and security claims.
Origin	Trust agent
Destination	Tracer
Interfaces	-
Step	(Step 2) Tracer validates the integrity of the device
Description	The integrity of the device is validated through secure boot firmware hash
Origin	Tracer
Destination	Tracer
Interfaces	-
Step	(Step 3.1) Tracer reported a failure in integrity of attestation agent
Description	If the validation of the integrity of the CMD fails, the tracer informs the attestation agent
Origin	Tracer
Destination	Attestation agent
Interfaces	-
Step	(Step 3.2) Tracer send data to attestation agent
Description	If the validation of the integrity of the CMD succeeds, the Tracer transmits the collected data to attestation agent
Origin	Tracer
Destination	Attestation Agent
Interfaces	-

Table 5.7: Flow for Story-XVIII: As a Bare Metal Tracer I want to protect the integrity and authenticity of the gathered data

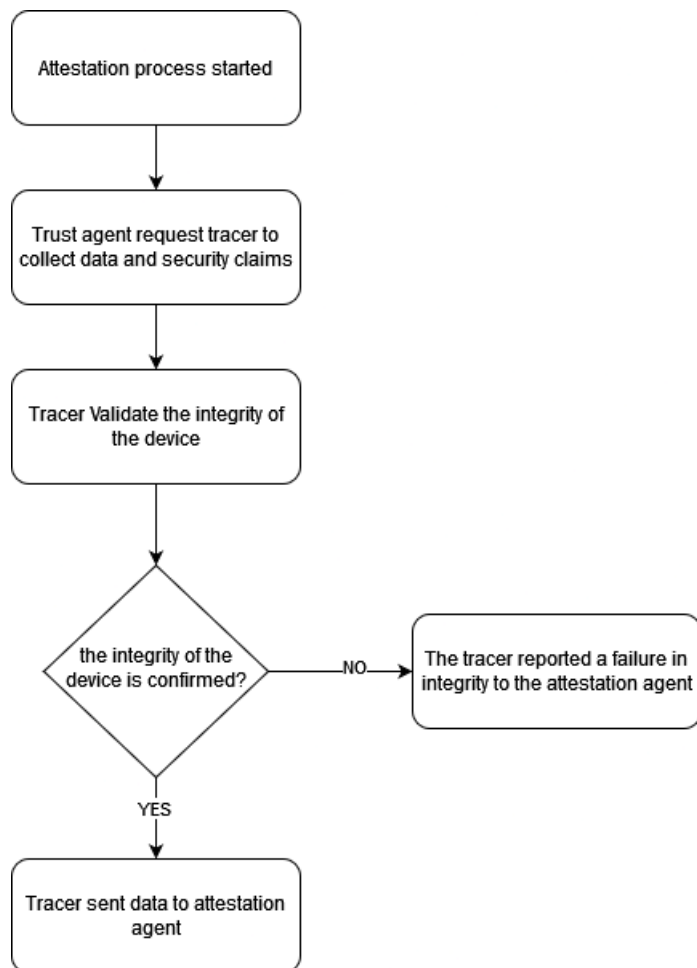


Figure 5.6: HS SPEC

Chapter 6

ENTRUST hybrid PUF-based Element Functionalities

After having described the set of cryptographic functionalities necessary for supporting the secure lifecycle of CMDs (Chapter 4), as well as the trusted extensions (Chapter 5) envisioned in the context of the ENTRUST, this chapter shifts its primary focus to the PUF-dependent functional specifications. These specifications will be articulated through two engineering stories, providing practical and relatable examples of their application. The first of those will be dedicated to the methodology and processes involved in the extraction of PUF-based random bit sequences. These sequences are pivotal for ensuring robust and continuous device authentication, leveraging the unique and unpredictable nature of the PUFs, for high- entropy bit-string generation. The second engineering story introduces the concept of swarm attestation which will utilize the PUF technology to dynamically attest a set of low-end CMDs. This is particularly important for such types of devices, which usually does not possess the adequate computational resources for typical attestation schemes. PUF-assisted swarm attestation can offer an efficient and effective approach supporting the integrity and trustworthiness of the entire (or part) fleet of CMDs with within a domain. In D4.2, in the context of the secure device onboarding, the swarm attestation scheme will be described thoroughly.

6.1 Engineering stories

Story-XIX: As a low-end CMD I want to be able to extract or to have access to random but reproducible bit sequences that I can use for my authentication utilizing high-end device.

Objective: The purpose of this engineering story is to provide a solid description of how a low-end device, by leveraging the unique PUF characteristics for generating high-entropy bit strings, will be equipped with advanced authentication capabilities despite its limited computation resources.

Motivation: CMDs, in most cases, are characterized by limited computational capabilities since they are designed to meet other requirements such as small footprint, minimal weight etc. In order to safeguard them against the always evolving threat landscape novel, cost effective and lightweight

approaches has to be adopted. Towards that direction, in the context of ENTRUST, an SRAM-based PUF implementation is considered to support the cryptographic operations needed to increase the cybersecurity posture of CMDs.

Building Blocks and Flow of Actions: The main ENTRUST components/services that are engaged in this engineering story are from the high-end device perspective, the key management extensions (as part of the Security Monitor (see Figure 3.6) and from the low-end device, the authentication agent and the SRAM-PUF instance. Any authentication request from the parties involved, activates the authentication agent from the low-end device side and a request for a challenge is forwarded to the high-end device which has in disposal the corresponding pool of challenges accompanied with the generated data created during the enrollment phase, in order to initiate the key generation process. Upon the reception of the aforementioned data the validity of the challenge is verified and in such a case, the SRAM-PUF utilizes the received challenge to generate a response and along with the helper data a key is generated, and a key exchange session is activated. If the generated key is identical with the one created during the enrollment phase, then Security Monitor of the high-end device authenticates the low-end CMD.

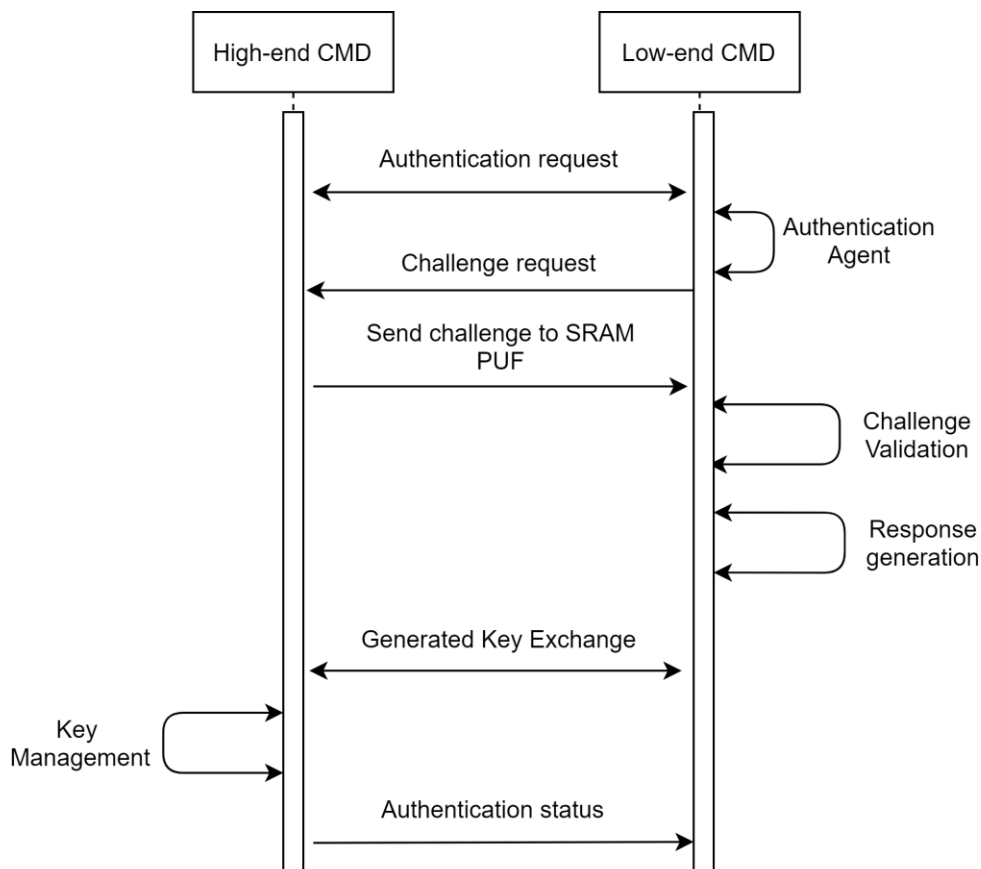


Figure 6.1: Authentication of low-end CDM utilizing high-end device leveraging high-entropy cryptographic primitives generated from a SRAM-based PUF instance.

Story-XX: As a domain manager I want to be able to attest a set or a subset of devices in the domain.

Objective: ENTRUST's main target is to establish attestable trust with groups of devices in large-scale dynamic and static networks, through designing a Scalable Aggregate Network Attestation called Swarm Attestation. Swarm Attestation aims to attest a group of devices in a scalable and efficient way instead of attesting each device separately.

Motivation: ENTRUST complex supply-chain ecosystems consist of large networks of heterogeneous medical devices as explained in Story-VI. These smart devices collaborate to provide all kinds of medical services and continuously monitor human health, such networks are called swarms. However, for such distributed services to be trusted, one needs to ensure that each device in the network behaves correctly. To prove the correctness of each device's software state, we must adopt remote attestation techniques that check the software integrity by detecting modified execution flows or data. Unfortunately, these schemes do not scale to many devices to be attested, such as a central Verifier attesting the entire collaborative network. Individually attesting each device in the network would induce a linearly growing overhead in communications and computations. Therefore, attesting large swarms requires a new and efficient approach.

Building Blocks: In general, swarm attestation schemes consider the attestation of two types of swarm networks: *static* or *dynamic* networks.

- **Static Swarms:** In a static swarm, the network is characterized by two main properties: the network is *static* and always *interconnected*, where two connected nodes are neighbors in the communication graph. An important property of a static swarm attestation is that each device can communicate only with its direct neighbors. In these schemes, each device attests its children and reports back to its parent the number of children that successfully passed the attestation protocol. In the end, an aggregated report with the total number of the devices successfully attested will be transmitted to the Verifier.
- **Dynamic Swarms:** In dynamic swarms, the network topology does not remain static, it changes even while the attestation protocol is executing. Overall, in dynamic attestation protocols, devices first share their respective "knowledge" with other devices, and then they use consensus mechanisms to agree on a piece of common knowledge about the whole network. ENTRUST will consider both Static and Dynamic Swarm networks to cover the ENTRUST use cases requirements and construct their respective Swarm Attestation protocols.

Swarm Attestation	
Step	(1) Request a Swarm Attestation from a group of connected medical devices, namely high and low-end devices
Description	Each parent, high-end device, is connected to a group of children, low-end devices, this constitutes a branch in the swarm. The verifier sends a challenge to be signed by the swarm

Origin	External Verifier
Destination	Swarm of medical devices
Interfaces	-
Step	(2) Creating branch signatures in the swarm
Description	Each mother medical device requests signatures from all its children's devices. The mother device then collects, verifies, and aggregates all its children's signatures. It also adds its signatures to the aggregation. This is a branch signature.
Origin	High-end devices
Destination	Low-end devices
Interfaces	-
Step	(3) Aggregation
Description	After creating branch signatures in the swarm, each branch sends its signature to the root device (Aggregator). The Aggregator aggregates all the branch signatures to form a swarm signature
Origin	High-end devices
Destination	Aggregator
Interfaces	-
Step	(4) Verification
Description	After creating a swarm signature, the aggregator sends the swarm signature to the verifier. The verifier verifies the signature and returns 0 or 1
Origin	Aggregator
Destination	Verifier
Interfaces	-

Table 6.1: Swarm Attestation

Chapter 7

Conclusions

The ultimate objective of ENTRUST is to deliver a comprehensive security management framework that ensures the integrity and trustworthiness of Connected Medical Devices (CMDs) throughout their entire lifecycle, from design to runtime phases. ENTRUST achieves this by leveraging trust-aware defense mechanisms rooted in the principles of Trusted Computing, employing innovative technologies and methodologies to safeguard CMD operations.

ENTRUST proposes a multifaceted Trusted Computing Base (TCB) architecture, adaptable to both low-end and high-end CMDs. This architecture incorporates Physical Unclonable Functions (PUFs) for lightweight devices and Trusted Execution Environments (TEEs) for more complex devices, ensuring a robust and scalable approach to security. The consortium's efforts have been focused on designing and implementing these customizable TCBs, which serve as the foundation for a secure CMD ecosystem.

The work detailed in this deliverable outlines several critical components and advancements, including:

- Cryptographic functionalities, including key management systems, enhanced authorization protocols, and secure communication mechanisms, to support the CMD lifecycle from onboarding to runtime operation and decommissioning.
- An attestation framework that combines formal verification with runtime attestation, to identify and monitor the trust boundaries of CMD operations.
- A secure enrollment and onboarding process utilizing verifiable credentials and presentations, ensuring the integrity and authenticity of CMDs within the network.
- A policy-based, stateful approach to software and firmware updates, ensuring secure and reliable maintenance and upgrade procedures.
- The establishment of Digital Twins and AI-based anomaly detection mechanisms, providing continuous monitoring and dynamic response capabilities to maintain CMD trustworthiness.

WP4 will continue to advance these initiatives, producing detailed designs and descriptions of the protocols, interfaces, and internal workings of the ENTRUST framework. Future deliverables will build on the foundations laid out in this document, further refining and implementing the security mechanisms essential for the holistic management of CMDs.

Overall, this deliverable represents a significant milestone in the development of ENTRUST's secure lifecycle management framework. By integrating innovative technologies and methodologies, ENTRUST is poised to set new standards in the trust and security of Connected Medical Devices, ensuring their safe and reliable operation across diverse healthcare environments.

Bibliography

- [1] Dsba releases ‘technical convergence discussion document’. <https://data-spaces-business-alliance.eu/dsba-releases-technical-convergence-discussion-document/>. Accessed: 2024-04-04.
- [2] Hardware Security Modules (HSMs) | Thales.
- [3] Support for TrustZone • Zynq 7000 SoC Technical Reference Manual (UG585) • Reader • AMD Technical Information Portal.
- [4] Cloud HSM - IBM Cloud, May 2016.
- [5] Bernard Aboba, Larry Blunk, John Vollbrecht, James Carlson, and Henrik Levkowetz. Extensible authentication protocol (eap). RFC 3748, Internet Engineering Task Force (IETF), June 2004.
- [6] Bernard Aboba, Dan Simon, and Pasi Eronen. Extensible authentication protocol (eap) key management framework. RFC 5247, Internet Engineering Task Force (IETF), August 2008.
- [7] Almudena Alcaide, Esther Palomar, José Montero-Castillo, and Arturo Ribagorda. Anonymous authentication for privacy-preserving iot target-driven applications. *Comput. Secur.*, 37:111–123, 2013.
- [8] Thaynara Alves and D. Felton. Trustzone: Integrated hardware and software security. 01 2004.
- [9] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP’07)*, pages 321–334. IEEE, 2007.
- [10] Karen H Brown. Security requirements for cryptographic modules. *Fed. Inf. Process. Stand. Publ*, pages 1–53, 1994.
- [11] Clemens Brunner, Ulrich Gellersdörfer, Fabian Knirsch, Dominik Engel, and Florian Matthes. DID and VC:Untangling Decentralized Identifiers and Verifiable Credentials for the Web of Trust. In *2020 the 3rd International Conference on Blockchain Technology and Applications*, pages 61–66, Xi’an China, December 2020. ACM.
- [12] Jan Camenisch, Manu Drijvers, Anja Lehmann, Gregory Neven, and Patrick Towa. Short threshold dynamic group signatures. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 401–423. Springer, 2020.
- [13] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd*

International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers, volume 9566 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015.

- [14] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 93–118, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [15] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001*. Springer, 2001.
- [16] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.
- [17] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [18] Chip-Hong Chang, Yue Zheng, and Le Zhang. A retrospective and a look forward: Fifteen years of physical unclonable function advancement. *IEEE Circuits and Systems Magazine*, 17(3):32–62, 2017.
- [19] Urbi Chatterjee, Vidya Govindan, Rajat Sadhukhan, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, Debashis Mahata, and Mukesh M Prabhu. Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database. *IEEE transactions on dependable and secure computing*, 16(3):424–437, 2018.
- [20] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
- [21] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
- [22] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, Internet Engineering Task Force (IETF), May 2008.
- [23] Mafalda Cortez, Apurva Dargar, Said Hamdioui, and Geert-Jan Schrijen. Modeling sram start-up behavior for physical unclonable functions. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2012.
- [24] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.
- [25] Peiwu Dai, Zulie Pan, and Yang Li. A review of researching on dynamic taint analysis technique. In *2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*, pages 118–123. Atlantis Press, 2018.
- [26] Alan DeKok. Extensible authentication protocol (eap) session-id derivation for eap subscriber identity module (eap-sim), eap authentication and key agreement (eap-aka), and protected eap (peap). RFC

8940, Internet Engineering Task Force (IETF), October 2020.

- [27] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances In Cryptology-EUROCRYPT 2004: International Conference On The Theory And Applications Of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 523–540. Springer, 2004.
- [28] Paul England, Andrey Marochko, Dennis Mattoon, Rob Spiger, Stefan Thom, and David Wooten. A foundation for trust in the internet of things. *Microsoft Corporation*, (Apr. 21, 2016), 17.
- [29] Nikos Fotiou, Vasilios A Siris, George C Polyzos, Yki Kortesniemi, and Dmitrij Lagutin. Capabilities-based access control for iot devices using verifiable credentials. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 222–228. IEEE, 2022.
- [30] Armknecht Frederik. A formal foundation for the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412, 2011.
- [31] P. Funk and S. Blake-Wilson. Extensible authentication protocol tunneled transport layer security authenticated protocol version 0 (eap-ttlsv0). RFC 5281, Internet Engineering Task Force (IETF), August 2008.
- [32] Jesús García-Rodríguez, Rafael Torres Moreno, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Towards a standardized model for privacy-preserving verifiable credentials. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 126:1–126:6. ACM, 2021.
- [33] Jesús García-Rodríguez and Antonio Skarmeta. A privacy-preserving attribute-based framework for IoT identity lifecycle management. *Computer Networks*, 236:110039, November 2023.
- [34] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 148–160, 2002.
- [35] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.
- [36] Harry Halpin. Vision: A Critique of Immunity Passports and W3C Decentralized Identifiers. In Thyla Van Der Merwe, Chris Mitchell, and Maryam Mehrnezhad, editors, *Security Standardisation Research*, volume 12529, pages 148–168. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.
- [37] Zeno Heeb, Onur Kalinagac, Wissem Soussi, and Gürkan Gür. The impact of manufacturer usage description (mud) on iot security. In *2022 1st International Conference on 6G Networking (6GNet)*, pages 1–4, 2022.
- [38] Hyunho Kang, Yohei Hori, Toshihiro Katashita, Manabu Hagiwara, and Keiichi Iwamura. Cryptographic key generation from puf data using efficient fuzzy extractors. In *16th International conference on advanced communication technology*, pages 23–26. IEEE, 2014.

- [39] Ben Lapid and Avishai Wool. Cache-attacks on the arm trustzone implementations of aes-256 and aes-256-gcm via gpu-based analysis. In *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*, pages 235–256. Springer, 2019.
- [40] Eliot Lear, Ralph Droms, and Dan Romascanu. Manufacturer usage description specification. RFC 8520, Internet Engineering Task Force (IETF), March 2019.
- [41] Ladislav Lhotka. JSON Encoding of Data Modeled with YANG. RFC 7951, aug 2016.
- [42] Wei Li. Evaluating the impacts of dynamic reconfiguration on the qos of running systems. *Journal of Systems and Software*, 84(12):2123–2138, 2011.
- [43] Wenhao Li, Yubin Xia, and Haibo Chen. Research on arm trustzone. *GetMobile: Mobile Computing and Communications*, 22(3):17–22, 2019.
- [44] Xi Jun Lin, Lin Sun, and Haipeng Qu. Insecurity of an anonymous authentication for privacy-preserving iot target-driven applications. *Comput. Secur.*, 48:142–149, 2015.
- [45] Jianghua Liu, Xinyi Huang, and Joseph K Liu. Secure sharing of personal health records in cloud computing: Ciphertext-policy attribute-based signcryption. *Future Generation Computer Systems*, 52:67–76, 2015.
- [46] Muqing Liu, Chen Zhou, Qianying Tang, Keshab K Parhi, and Chris H Kim. A data remanence based approach to generate 100% stable keys from an sram physical unclonable function. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2017.
- [47] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. A soft decision helper data algorithm for sram pufs. In *2009 IEEE international symposium on information theory*, pages 2101–2105. IEEE, 2009.
- [48] Hemanta K Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In *Cryptographers’ track at the RSA conference*, pages 376–392. Springer, 2011.
- [49] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2111–2128, 2019.
- [50] Robert H Morelos-Zaragoza. *The art of error correcting coding*. John Wiley & Sons, 2006.
- [51] Georgios M Nikolopoulos and Marc Fischlin. Quantum key distribution with post-processing driven by physical unclonable functions. *Applied Sciences*, 14(1):464, 2024.
- [52] Arttu Paju, Muhammad Owais Javed, Juha Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. Sok: A systematic review of tee usage for developing trusted applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–15, 2023.
- [53] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.
- [54] Akshay Parihar, Jigna B. Prajapati, Bhupendra G. Prajapati, Binti Trambadiya, Arti Thakkar, and Pinalkumar Engineer. Role of iot in healthcare: Applications, security privacy concerns. *Intelligent*

Pharmacy, 2024.

- [55] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*, page 111–126, Berlin, Heidelberg, 2016. Springer.
- [56] Soumyakant Priyadarshan. A study of binary instrumentation techniques. 2019.
- [57] Kai Rannenberg, Jan Camenisch, and Ahmad Sabouri, editors. *Attribute-based Credentials for Trust: Identity in the Information Society*. Springer, 2015.
- [58] Dominik Arne Rebro, Stanislav Chren, and Bruno Rossi. Source code metrics for software defects prediction. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, SAC'23*, page 1469–1472, New York, NY, USA, June 2023. Association for Computing Machinery.
- [59] B Indira Reddy and V. Srikanth. Review on wireless security protocols (wep, wpa, wpa2 wpa3). *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pages 28–35, 07 2019.
- [60] Alfredo Rial and Ania M. Piotrowska. Security analysis of coconut, an attribute-based credential scheme with threshold issuance. *IACR Cryptol. ePrint Arch.*, page 11, 2022.
- [61] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret: Theory and applications of ring signatures. *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 164–186, 2006.
- [62] MAES Roel. Physically unclonable functions: Constructions, properties and applications. *Katholieke Universiteit Leuven, Belgium*, pages 148–160, 2012.
- [63] Roberto Román, Rosario Arjona, and Iluminada Baturone. A lightweight remote attestation using pufs and hash-based signatures for low-end iot devices. *Future Generation Computer Systems*, 148:425–435, 2023.
- [64] Shruti Sakhare and Dipti Sakhare. A review—hardware security using puf (physical unclonable function). In *ICCCE 2019: Proceedings of the 2nd International Conference on Communications and Cyber Physical Engineering*, pages 373–377. Springer, 2020.
- [65] Jose Luis Canovas Sanchez, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Integration of anonymous credential systems in iot constrained environments. *IEEE Access*, 6:4767–4778, 2018.
- [66] Jose Luis Canovas Sanchez, Jorge Bernal Bernabé, and Antonio F. Skarmeta. Towards privacy preserving data provenance for the internet of things. In *4th IEEE World Forum on Internet of Things, WF-IoT 2018, Singapore, February 5-8, 2018*, pages 41–46. IEEE, 2018.
- [67] D. Simon, B. Aboba, and R. Hurst. The eap-tls authentication protocol. RFC 5216, Internet Engineering Task Force (IETF), March 2008.
- [68] Dan Simon and Bernard Aboba. Guidance for authentication, authorization, and accounting (aaa) key management. RFC 4962, Internet Engineering Task Force (IETF), July 2007.

- [69] Manu Sporny, Dave Longley, David Chadwick, and Orie Steele. Verifiable credentials data model v2.0. *W3C, W3C Recommendation*, 2024.
- [70] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Sgx-step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, pages 1–6, 2017.
- [71] Wikipedia. Trusted Computing Base. https://en.wikipedia.org/w/index.php?title=Trusted_computing_base. Accessed 10-Dec-2023.
- [72] Johannes Winter. Experimenting with arm trustzone—or: How i met friendly piece of trusted hardware. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1161–1166. IEEE, 2012.
- [73] Kan Xiao, Md Tauhidur Rahman, Domenic Forte, Yu Huang, Mei Su, and Mohammad Tehranipoor. Bit selection algorithm suitable for high-volume production of sram-puf. In *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*, pages 101–106. IEEE, 2014.
- [74] Xiaolu Zhang, Oren Upton, Nicole Lang Beebe, and Kim-Kwang Raymond Choo. Iot botnet forensics: A comprehensive digital forensic case study on mirai botnet servers. *Forensic Science International: Digital Investigation*, 32:300926, 2020.