TU WIEN Informatics

On Hardware-based Security in Embedded Systems

Evaluating potential use of secure hardware in C-ITS stations

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Gerhard Hechenberger, BSc.
Matrikelnummer 01326157

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Mitwirkung: Dipl.-Ing. Herbert Fuereder

Wien, 21. Jänner 2020

Gerhard Hechenberger   Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Informatics

On Hardware-based Security in Embedded Systems

Evaluating potential use of secure hardware in C-ITS stations

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Gerhard Hechenberger, BSc.
Registration Number 01326157

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Dipl.-Ing. Herbert Fuereder

Vienna, 21st January, 2020

Gerhard Hechenberger | Edgar Weippl

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Gerhard Hechenberger, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Jänner 2020

Gerhard Hechenberger

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ich möchte mich bei meinem Betreuer Prof. Weippl für die Möglichkeit bedanken, diese Arbeit umsetzen zu können, als auch für seine Anleitung während dieser Zeit. Bei meiner Chefin Karin und meinen Arbeitskollegen Herbert and Thomas möchte ich mich ebenfalls bedanken, für ihre Unterstützung in organisatorischen sowie technischen Belangen. In meiner Zeit als Werkstudent bei Siemens habe ich wirklich viel von ihnen gelernt. Ein spezielles Danke geht auch an meinen langjährigen Freund Luca, für seine andauernde Unterstützung in Zeiten stressiger Semester und anstrengenden Prüfungsvorbereitungen. Nicht zuletzt möchte ich meinen Eltern Martin und Brigitta danken, für ihre vorbehaltlose menschliche und finanzielle Unterstützung, die mir so viele Möglichkeiten eröffnet hat. Ohne sie wäre diese Arbeit nie möglich gewesen.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to thank my adviser Prof. Weippl for the possibility of creating this work and his guidance, as also my boss Karin and my colleagues Herbert and Thomas for their support on organizational and technical means. I really learned a lot from them during my time as a working student at Siemens. A special thanks also goes to my long friend Luca for his enduring support during stressful semesters and tough exam preparations. Finally, I want to thank my parents Martin and Brigitta, for their unconditional support on social and financial matters which offered me so many opportunities. Without them, this work would have never been possible.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Collaborative Intelligent Transport Systems (C-ITS) Stationen sind aktuell aufkommende Internet of Things (IoT) Geräte im Bereich der Verkehrsinformation und -kontrolle. Überwiegend befestigt an Straßenkreuzungen werden sie genutzt, um Vehicle-to-everything (V2X) Nachrichten zu senden und weiterzuleiten und über unterschiedlichste Kanäle zu kommunizieren. Durch den Stellenwert der Safety bei Verkehrsinfrastruktur ist hier Security sehr wichtig. Zusätzlich sind Geräte, die den Angreifern physikalischen Zugang ermöglichen, speziell exponiert und benötigen Security-Maßnahmen, die durch Hardware unterstützt werden.

In unserer Arbeit zur Verbesserung der Security in C-ITS Stationen der nächsten Generation analysieren wir zuerst basierend auf IEC 62443 System- und Service-Anforderungen und erstellen eine Threat and Risk Analysis (TRA). Danach untersuchen wir die Verfügbarkeit und Funktionalität von Security-Hardware-Modulen, um ein in ein ganzheitliches System-Security Konzept eingebettetes, Hardware-unterstütztes Key-Management zu entwickeln. Durch die Implementierung auf einem NXP i.MX8QXP Evaluation Kit erreichen wir Einblick in die Ausgereiftheit der Software, den Entwicklungsprozess sowie potenzielle Stolperfallen und Probleme der sicheren System-Entwicklung.

Unser entwickeltes Konzept erfüllt unsere Anforderungen und zeigt signifikante Verbesserungen in der TRA. Allerdings muss durch die Nutzung eines Trusted Execution Environments (TEEs), wie erwartet, bei der Verschlüsselungs-Performance im Vergleich zu OpenSSL ein Rückgang um Faktor 30 für kleine Datenmengen und 2.4 für große Datenmengen akzeptiert werden. Während unserer Untersuchung konnten wir mehrere Implementierungsmängel in der verfügbaren Software entdecken, die sowohl Funktionalität als auch Security betreffen. Für einige davon bieten wir Lösungsvorschläge an und beschäftigen uns schlussendlich noch mit den nötigen Schritten zur Übernahme in den Produktiv-Betrieb.

Schlüsselwörter — *Collaborative Intelligent Transport Systems, RSU, Hardware security modules, Embedded systems security, Secure boot, Secure key management, U-Boot, OP-TEE, i.MX8*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Collaborative Intelligent Transport Systems (C-ITS) stations are recently upcoming Internet of Things (IoT) devices in the domain of traffic information and control. Meant to be deployed at intersections, they are working as highly heterogeneous routing devices to send and relay Vehicle-to-everything (V2X) messages. As traffic infrastructure poses safety implications, security is particularly important here. Additionally, devices which give physical access to attackers are especially exposed and require security measures which need the underlying hardware's support.

In our work to improve security in the next generation of C-ITS stations, we first analyze for system and service requirements based on IEC 62443 and conduct a Threat and Risk Analysis (TRA). We then survey the market about available secure hardware modules and its provided functionalities, to be able to set up a hardware-supported secure key management embedded in a full system-security concept. Implementing this using an NXP i.MX8QXP evaluation kit lets us gain further insights about its software maturity, the development process and potential pitfalls and problems of secure systems engineering.

Our created concept satisfies the given requirements and shows to significantly improve the TRA. However, as expected due to the usage of a Trusted Execution Environment (TEE), encryption performance suffered from a drop of factor 30 for small files to a drop of about factor 2.4 for big files in a comparison against OpenSSL. During our research, we also discover multiple implementation shortcomings of the provided software concerning functionality and security, propose fixes and summarize the steps needed to move to production.

Keywords *— Collaborative Intelligent Transport Systems, RSU, Hardware security modules, Embedded systems security, Secure boot, Secure key management, U-Boot, OP-TEE, i.MX8*

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **xi**

Abstract **xiii**

1 Introduction **1**
1.1 Motivation and Problem Statement . . . 1
1.2 Aim of the Work . . . 3
1.3 Methodological Approach . . . 4
1.4 Structure of the Work . . . 5

2 Background **7**
2.1 C-ITS Standards and Security . . . 7
2.2 Embedded Systems Security . . . 8
2.3 Secure Hardware . . . 9
2.4 Boot Flow . . . 12
2.5 Threat Modeling for CPS . . . 15

3 Analysis **19**
3.1 Frameworks . . . 19
3.2 Requirements Analysis . . . 22
3.3 Threat and Risk Analysis . . . 24
3.4 Market Survey . . . 29
3.5 Evaluation . . . 35

4 Concept and Implementation **39**
4.1 Concept Requirements . . . 39
4.2 Key Management Approaches . . . 40
4.3 Full System Concept . . . 44
4.4 NXP iMX8 Boot & OP-TEE Setup . . . 46
4.5 System Authentication by NXP . . . 57
4.6 Key Management by TA . . . 59

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 **Evaluation** 69
5.1 Security 69
5.2 Performance 75
5.3 Services 77
5.4 Moving to Production 78
5.5 Development Takeaways 79

6 **Summary** 83
6.1 Requirements and Analysis 83
6.2 Concept and Implementation 83
6.3 Findings 84
6.4 Future Work 85

List of Figures 87

List of Tables 89

List of Listings 91

Acronyms 93

Bibliography 97

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

During recent years, the Internet of Things (IoT) emerged and internet-connected embedded systems are conquering the world in constantly rising numbers. Information sharing turned out to be an effective way to improve our increasingly dependent world. When it comes to transportation, passing messages from car to car or to their surroundings can highly improve safety, traffic flow and subsequently fuel efficiency, and provide improvements not only in current vehicles but also to support autonomous driving in the long run. To make this possible and support current cars with additional information, within the scope of Collaborative Intelligent Transport Systems (C-ITS) Vehicle-to-everything (V2X) communication is currently in the standardization process by established institutions, including the Institute of Electrical and Electronics Engineers (IEEE) and European Telecommunications Standards Institute (ETSI), in cooperation with international partners[1]. Devices compatible to the first standard publications are currently under heavy development throughout the transportation industry, and recently VW presented the first consumer product including this technology, their new Golf 8[2].

Due to the basic communication structure, also the security requirements for the communication protocol are high and already standardized. The V2X communication structure is designed as a mesh-network, omitting a central point of authority. To keep it fast and simple, no sessions are used and information is directly broadcasted (and eventually received) by the various devices in near distance. This leads to the need of signing all messages sent to and verifying all messages received from participants, to provide message integrity. It can either be realized using WiFi (802.11p) or cellular (LTE-V2X,

[1] https://www.etsi.org/technologies-clusters/technologies/automotive-intelligent-transport

[2] https://www.forbes.com/sites/samabuelsamid/2019/10/28/volkswagen-includes-nxp-v2x-communications-in-8th-gen-golf

in future 5G) networks. But there exist not only devices integrated in vehicles to provide information about their environment, the transportation infrastructure also needs to communicate with them. For this application, a dedicated type of C-ITS stations, so called Road Side Units (RSUs), are also currently under development. They are intended to be deployed at intersections and various other regions of interest, to communicate with the passing cars, informing them about their surroundings and supporting them with additional information, while being remotely administered and connected over the Internet[3].

As they are basically embedded systems, designed as highly heterogeneous routing devices connected to the Internet, they have many different interfaces and run more or less typical IoT services. This requires a sophisticated and holistic security concept, as the effects of malicious manipulations can be catastrophic for traffic participants' safety, especially with the future goal of autonomous driving. The first generation of RSUs is currently used for development projects and the design for the second generation is on its way, planned to be used on a much bigger scale throughout Europe and the US.

The market of V2X devices is recently gaining attention and poses new requirements and services, while a lot of manufacturers are still struggling with securing their traditional IoT devices [1, 2]. The security problems in such devices are quite unique and promoted by resource constraints and cheap manufacturing cost targets. This often means barely or no updates at all. Additionally, these IoT devices are often operated out-of-sight and with potential physical access for attackers. This leads to unique threats targeted directly at the hardware level and therefore also demand countermeasures on hardware level, as software here is not enough any more. This work tries to build some knowledge about applied hardware-base security in this market by examining the following problems:

- What secure hardware suited for use within C-ITS stations is available and what functionality does it provide?
- Which typical C-ITS station services may gain security improvements by using such secure hardware and how can it be integrated into a system security concept?

A lot of research on security in embedded systems has been done recently. Cornerstones of a security concept are typically secure boot and storage [3]. To implement each of them, as also for other services, one needs some kind of secret keys or certificates. As these secrets now cannot be protected by a password supplied by a user, they have to be protected and managed in a different way without depending on external actions. Therefore, the hands-on part of this thesis will examine:

- What possibilities exist to securely do the key management in C-ITS stations using hardware support and what are their advantages and disadvantages?
- By using one of the possibilities of the last question, how well does the implementation and integration in a secure system work and where are the potential pitfalls?

[3]https://new.siemens.com/global/en/products/mobility/road-solutions/connected-mobility-solutions/sitraffic-vehicle2x.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The last question also aims at building a solid understanding of the system and security concept, which shall later be used for further work in this area.

1.2 Aim of the Work

The aim of this work is to enable and support security improvements by secure hardware in next-generation C-ITS stations. This is achieved by multiple steps.

1.2.1 Market Survey

A market survey is conducted to build knowledge about available stand-alone and System on Chip (SoC) integrated secure hardware modules and to be able to examine their applicability in the upcoming next generation of C-ITS stations. It also aims at giving some guidance on potential security improvements and linking them to a usage in V2X communication specific services.

1.2.2 PoC Implementation

In engineering, unknown problems tend to constantly arise during the design and implementation process of new devices. Therefore, one of the surveyed secure hardware modules is used to create a Proof of Concept (PoC), which shows how key management can be improved by using secure hardware (due to the limited scope, only one PoC was feasible). This fully aims at the applied security domain – to get an idea where the challenges are if one plans to integrate such devices in their products and at creating a well-documented basis for future integration and possible enhancements in other projects. The final evaluation shows us not only the security implications of the solution, but it also considers engineering metrics like portability, estimated implementation effort and software and documentation quality.

1.2.3 Target Audience

The targeted audience is expected to have a broad knowledge on the computer science and information security domain, such as common terminology, cryptography, widespread threats and basic attacks. They should also have some background on hardware and its low-level software to be able to completely understand the topics in this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.3 Methodological Approach

The used methodology will be outlined in the following sections.

1.3.1 Literature Review

As a solid base, a broad literature review is conducted. It builds some general knowledge about C-ITS devices and their security challenges, and shows us the state-of-the-art of embedded systems security in the domain of Cyber-Physical Systems (CPS) and the IoT. Threat modeling challenges for such devices and existing secure hardware are also addressed. On the basis of this review, the market survey is initiated.

1.3.2 Market Survey

Existing stand-alone and SoC integrated secure hardware modules targeted more or less for automotive usage are gathered and examined for their provided features, interesting properties and special functionality to improve the security of C-ITS stations.

1.3.3 System Analysis

To evaluate their potential impact on the system security, a requirement analysis and subsequently a Threat and Risk Analysis (TRA) are created for the examined system services. This is done in the context of a later integration in a system-wide standard currently developed and used for Industrial Automation Control Systems (IACS), the IEC 62443 [4], and picking an established and suitable TRA framework which can be used for embedded systems, namely Microsoft's STRIDE and DREAD. How they may be modified to fit the world of CPS is also briefly covered.

Knowing the requirements and TRA outcome, we now analyze the potential applicability to the system services and their potential security improvements. This shows, which threats can be addressed and to which extent they can be mitigated by the use of secure hardware to support further development decisions.

1.3.4 PoC Implementation

To improve the security of the key management in C-ITS stations and assist a PoC implementation, various different concepts based on secure hardware of our preceding analysis are created. They get rated and one is picked for a hands-on PoC. The implementation is done using security best-practices and focusing on the applied security aspect, which is also considered in the final evaluation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.4 Structure of the Work

This chapter outlines the motivation, formulates the existing problems to solve, and explains the used methodology. Some related work will be presented in the next chapter, chapter 2. It also includes an overview of common standards in the scope of this work and various general concepts, which will be used in the following chapters. The market survey is done in chapter 3, including a preceding explanation of the used framework for system analysis and the TRA. The evaluation of this analysis concludes this chapter. Based on this evaluation, different approaches to improve the system's key management are created in chapter 4 and a security concept is developed after rating and choosing one of them for a PoC implementation. Chapter 5 covers the evaluation of the concept in terms of security, performance and miscellaneous things worth noting. The last chapter of this work, chapter 6, summarizes the completed work and highlights open questions to be addressed in the future.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

To create a solid basis of knowledge this thesis is built on, the following sections will list and explain related work and common concepts of C-ITS and embedded systems in the context of security. The focus will be on hardware concepts and features, as also modeling methods for threats.

2.1 C-ITS Standards and Security

As this work exists in the context of C-ITS, some background on this topic is vital. Section 1.1 already introduced some high-level basics on V2X technology and C-ITS stations. On a lower level, in [5], Toetzl provides some basics for the used ETSI ITS-G5 standard, as also the used 802.11p WiFi layer described in ETSI ES 202 663 [6]. The communication is based on ad-hoc networks and does not support sessions for connections, therefore it needs to be embedded in a well-suited security concept. This results in the fact that every message has to be signed, requiring a sophisticated Public Key Infrastructure (PKI) as also described in [5]. Additional requirements on how the resulting keys and certificates have to be managed are specified by ETSI TS 102 940 [7], also including the C-ITS station communication security architecture and various defined security services on top of them. The Intelligent Transport Systems (ITS) station security management is also specified, as well as guidelines for establishing trust on such devices.

Various publications target the standards and their state of implementation [8, 9]. They give an overview over the standardization bodies involved and highlight, that due to the network structure, “in these environments, security is considered in design and implementation since compromised vulnerabilities in one vehicle can be propagated to other vehicles”. However, there is always room for improvement. In [10], Gafencu et al. analyze special requirements for security (due to the very dynamic networks caused by moving nodes, the huge network scale, non-uniform distributed nodes and network

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

coverage) in ITS devices. Differences and similarities to the IoT are highlighted, and some measures for upcoming security improvements are proposed.

A recent development is also the aspired standardization of the European C-ITS market[1], which was pushed forward by the European Union (EU) in 2019 by a delegated directive [11], that also covers additional security requirements in ANNEX 3. Although the directive was rejected by some of its member states[2], the controversial topics did not include the specified security requirements. Therefore, we will likely see similar requirements in any future effort of standardizing the European market, although this will take additional time. In the meantime, a lot of big companies partnered up to adhere to many aspects of the rejected delegated directive due to the lack of other standards, as there exists a kind of consensus on the need of standards in this area.

2.2 Embedded Systems Security

Given the fact that connected devices are constantly getting cheaper and are used for a growing number of applications, the topic of embedded systems security is gaining more and more momentum in recent years.

An overview of this is given by Papp et al. in [12]. They cover various threats and vulnerabilities of embedded systems, including their own attack taxonomy. According to that, the most successful approach by attackers is based on an internet-facing device, where either a vulnerable web application is running, the access control/authentication is weak or some other application with a basic programming error exists. Hardware is very rarely targeted, mostly it's applications or the firmware/Operating System (OS) itself. They want to support structural analysis and design of embedded systems, which is especially important as the physical protection of such devices is often hard to ensure and security highly affects their dependability and safety. In [13], Ravi et al. also examine general attack types and survey "tamper resistant designs", which shall help to prevent tampering with the device by either preventing, recovering or detecting attacks. This can be supported not only by software, but also by hardware. Another type of vulnerabilities which got a lot of attention lately is hardware-rooted. Mostly using Side-Channel Attacks (SCAs) to extract information from computer systems, these vulnerabilities hold whole new challenges for designing secure systems. In [14], Fournaris et al. are surveying potential microarchitectural attacks and outline some approaches to mitigate them. Except the original Rowhammer paper [15] they also mentioned, the Meltdown [16] and Spectre [17] vulnerabilities were gaining a lot of attention in 2018. Extending this research, the ZombieLoad [18] vulnerability was published in the following year, and also Rowhammer evolved to RAMBleed [19].

However, long before, the scientific community observed that the unique challenges of embedded systems security need a suitable design process, adapted models and new

[1] https://ec.europa.eu/transport/themes/its/c-its_en
[2] https://agenceurope.eu/en/bulletin/article/12291/30

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

architectures. A lot of researchers were writing about design challenges in embedded systems and naming security a new dimension, that has already had to be considered throughout the whole development phase of embedded systems engineering [20, 21, 22]. However, years later, the state of security in embedded systems was still “a mess”, as Viega et al. in [2] called it. They confirm the need of a different design approach from their data, but also acknowledge, that implementing security measures for new code is tough, for legacy code hard. They also mention, that due to the infrequent updates, these devices barely benefit from vulnerability management measures, what can be also seen in the survey of Pescatore et al. [1]. More about CPS design challenges was published aside a workshop of the Cyber Security Research Alliance (CSRA) [23]. The participants stress the role of security in such embedded systems, as the implications of failures may impair physical safety up to loss of life. They give recommendations on how to achieve this goal, even down to the supply chain.

When it comes to C-ITS and connected cars, as part of the IoT they face similar problems and challenges [24]. In [25], Roudier et al. bring the model-driven security approach to the automotive world, supported by a study during the E-safety Vehicle Intrusion Protected Applications (EVITA) project. And as cars get smarter and smarter, Markantonakis et al. in chapter 12 of [26] examine attackers and attack paths including threats, risks and privacy aspects of smart, embedded, automotive platforms. Soja in his white paper [27] outlines a different aspect – the application of standards to create good implementations. He argues, that due to the increasing connectivity, no safety without security exists any more, and therefore the industry must develop standards and design with the possibility of attacks in mind. As he lays the focus on the application, he outlines various security mechanisms, starting from secure flash programming, building a Chain of Trust (CoT) for secure boot and using dedicated cipher engines for external memory security. Other mechanisms, such as the importance of a good key management as well as general security best practices were examined by chapter 6 in [26]. Chapter 18 also highlights the advantages of well evaluated security implementations and methodology like Common Criteria (CC) to achieve that.

2.3 Secure Hardware

Already before embedded systems conquered the world, security had been a problem which could not fully be solved by software. Over decades, the architecture of today’s computers was extended and modified to fit new requirements, resulting in a sub-optimal design for security. Nowadays, hardware can provide an additional layer of security which software cannot, located even below the OS and providing tampering protection, defense against malicious software and side-channel mitigation. As Cheruvu et al. in [28] state, the important parts to achieve are the creation of a device identity, protected boot and protected storage. This can be achieved in various ways, leveraging special hardware as Trusted Platform Modules (TPMs) or a Trusted Execution Environment (TEE) supported by software. The following paragraphs will now make a short excursion to existing hardware security building blocks.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Sanchez-Reillo et al. in [29] analyzed how to use "security hardware modules" to defeat upcoming security holes hardly addressable by software. Over the years, various hardware modules were established in the security domain, including TPMs, Mobile Trusted Modules (MTMs) and Hardware Security Modules (HSMs). Their basics are described in-depth in chapters 4 and 17 of [26]. TPMs, whose development and standardization is driven by the Trusted Computing Group (TCG), are used to provide cryptographic operations as also a Root of Trust (RoT) for storage, reporting and integrity measurement. In [30] also the Public Key Cryptography Standard #11 (PKCS#11) is mentioned, a industry standard API for cryptographic hardware. As requirements for mobile devices differ, MTMs take the place of a mobile version of TPM. HSMs take a similar role, they typically provide cryptographic functions and storage for cryptographic keys, including also tampering protection. In [31], Karter et al. evaluate various TPM security features and formulate key benefits, as there is the confidence in the platform, platform-bound data, owner privacy and control as also secure boot. Especially in the scope of embedded systems, Wolf et al. in [32] examine the usage of HSMs, as they argue that their hardware layer is particularly exposed to physical attacks where tamper-protected hardware helps to protect critical information. They also highlight, that hardware measures cannot help neither in case of software vulnerabilities nor in fundamentally flawed designs.

A survey on current crypto-processors and their applications was recently published by Sau et al. [33]. They give an overview of hardware related vulnerabilities and countermeasures and take a look at various TEE methods and multiple approaches for secure boot. They conclude that trusted boot, TEE and secured storage are the main features for reasonable system security. To generalize such approaches, Löhr et al. in [3] introduce security patterns for secure boot and secure storage, both important basic trusted computing concepts, aiming to enhance security by using a combination of trusted hard- and software components. Secure boot is the requirement for most system security solutions, whereas secure storage is vital for application-level security.

We have already heard about trusted computing and TEEs, which are used to support the design of complex and secure systems. A definition of a TEE is given by Sabt et al. in [34]. They describe it as an isolated and "tamper-resistant processing environment", in which applications can be securely executed. They take a look at the existing Advanced RISC Machine (ARM) TEE implementation, the ARM TrustZone[3], and define some general TEE building blocks. Due to the huge distribution of ARM processors in the mobile world, the ARM TrustZone is the de-facto standard there. Its architectural design is described in [35]. Some early experiments using the TrustZone were documented by Winter et al. in [36]. Their focus lies at system-level development on inexpensive TrustZone-enabled hardware, also possible in class-room settings. The general design and implementation of embedded systems based on the TrustZone is examined by Yan-Ling et al. in [37] to eliminate security weaknesses and enhance safety practices. They propose a multi-policy access control mechanism with a secure reinforcement method, building on that their prototype achieves a rational combination of secure OS and trusted hardware. An

[3] https://developer.arm.com/ip-products/security-ip/trustzone

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

overview of the architectural features and use cases is given by Ngabonziza et al. in [38]. They discuss details of different ARM architectures with TrustZone support, review their hardware and software implementation and conclude that they provide great flexibility, while avoiding the scenario of an all-mighty black box as a system controller. Quite recently, Pinto et al. also created a comprehensive survey on TrustZone [39]. Realizing that recent activities have significantly advanced its state, they conducted an in-depth study and analyzed the most relevant system weaknesses, aiming to help researchers and developers to familiarize with the concepts. Looking forward, they believe that the IoT has the potential to yield high-impact contributions and increases the awareness of TrustZone as a powerful security building block for embedded systems. As one of the new improvements, Zhao et al. in [40] propose a private user data protection mechanism based on identity authentication. Using this, it is possible for Trusted Applications (TAs) to perform identity authentication on normal world applications calling it, and therefore prevent potential user data leakage. Their results show that their solution can provide effective countermeasures.

There exist numerous software implementations on top of the ARM TrustZone, created for different purposes. In an effort to make development of TAs for TEEs easier, the GlobalPlatform (GP) group created a standard for TEE core APIs [41]. Hence, the TAs can become independent of the underlying TEE implementation. Adhering to the GP standard, Open-TEE was created mostly as a research project and for development support [42]. It resembles a virtual, hardware-independent TEE implemented in software, which developers can use to develop and debug their TAs on. When the development finished, the source can easily be compiled for any other hardware TEE using the same standard. Such an other TEE implementation is the Open Platform Trusted Execution Environment (OP-TEE)[4], supported by Linaro. It is open-source and has an extensive documentation online [43]. Therefore, it is gaining more and more attention now. Nehal et al. in [44] examined, how to secure IoT applications with OP-TEE. Due to its open-source character, they see OP-TEE as an important step to take security to every platform and as the future of securing IoT devices at hardware level. How to develop secure services for IoT devices with OP-TEE was also a topic of Göttel et al. in [45]. They implemented a key-value store and examined its performance and usability in contrast to the native secure storage implementation. Unsurprisingly, using secure storage goes hand in hand with a significant performance overhead.

In the C-ITS domain, the EVITA[5] project aims at "secure and trustworthy automotive on-board IT systems", as stated in their first publication [46]. Their approach of the security requirement analysis together with hard- and software design shall serve as a basis for future projects. One of their last publications [47] describes typical features of such systems and how secure keys should be handled in this context. They provide an analysis of their approach and also compare it to other cryptographic modules.

As security is not only about cryptography, there also exist other modules whose features

[4] https://www.op-tee.org/
[5] https://www.evita-project.org

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

can be leveraged to improve security. One of these are embedded Multi-Media Card (eMMC) modules, which can be more than a simple memory chip. They usually provide separate hardware partitions for boot and since its standard version 4.4 [48] also an Replay-Protected Memory Block (RPMB) partition. This can be leveraged against replay attacks, as Zilberstein et al. in [49] together with other eMMC features describe. It was also examined by Reddy et al. in [50], to protect secure data on mobile devices. Their proposed implementation guarantees secure storage against hacking attacks. Another quite interesting thing, though not ready for use yet, are Physically Unclonable Functions (PUFs). Also described in chapter 19 of [26], they can be used to uniquely identify and authenticate devices.

However, there is no hundred percent security, as also the recent publication of the TPM-FAIL attack by Moghimi et al. shows [51]. Hardware modules may introduce their own attack vectors and vulnerabilities. Some of the listed papers already include shortcomings of the examined modules and others focus solely on attacks, as Murdock et al. in [52] with the very recently discovered possibility of compromising Intel's TEE implementation via undervolting (named Plundervolt) or as Chen et al. in [53], where they describe a downgrade attack on the ARM TrustZone. They exploit reused verification keys and lacking rollback protection to achieve this. As some of these shortcomings are rooted in replay protection, RPMB may be leveraged to counter them. And not only technical shortcomings hinder the thriving of hardware security usage – as Batina et al. in [54] show on the example of TEEs, they still have unsolved technical issues on their own and licensing issues complicate their adoption. Keeping this in mind, however, as this section shows, we still have a lot of existing hardware security measures to improve the security state of embedded systems.

2.4 Boot Flow

As we have seen in section 2.3, secure boot is considered a really important primitive of ensuring the security of computing devices. More on that in section 2.4.6, but first, here some background on the general boot process. An abstract schematics is given in figure 2.1, where one can see how a standard personal computer usually starts up to its using state.

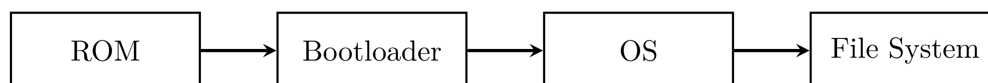

Figure 2.1: Standard boot flow

In embedded systems, this flow is highly dependent on the hardware platform used. However, the ARMv8 platform aims to define a standardized secure boot flow for all its processors, which system design is explained in the official documentation [55]. The ARMv8 AArch64 boot path can be seen in figure 2.2, its stages (or boot levels BL1,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

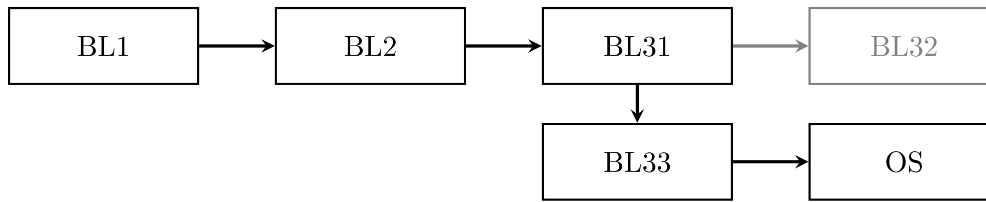

Figure 2.2: ARMv8 AArch64 boot flow

Non-trusted world		Trusted world	
Level	**Software**	**Software**	**Level**
EL0	RA	TA	S-EL0
EL1	ROS	TOS	S-EL1
EL2	Hypervisor	–	–
–	–	Secure monitor	EL3

Table 2.1: ARMv8 privilege execution levels (adapted from [56])

BL2, BL3-1, BL3-2 (optional) and BL3-3) are discussed in the following sections. The corresponding privilege levels used throughout the explanations are given in table 2.1.

2.4.1 BL1: AP Trusted ROM

This denotes the usually fixed boot code in Read-Only Memory (ROM) and is typically supplied by the vendor of a SoC. It starts the boot path with the highest privilege level EL3 and performs various architectural and platform initialization tasks, such as initializing the CPU and its control registers, as also the console, watchdog, Memory Management Unit (MMU) and interconnect. It prepares the system state for loading the next stage, BL2.

2.4.2 BL2: Trusted Boot Firmware

This denotes the so called Secondary Program Loader (SPL). It runs in S-EL1 and continues the architectural and platform initialization tasks. It also loads the subsequent images into Random-Access Memory (RAM) and hands over control to BL31.

2.4.3 BL31: EL3 Runtime Software

This stage is solely executed in trusted SRAM on level EL3. A reference implementation according to the Trusted Board Boot Requirements [57] of ARM is available, the ARM Trusted Firmware (ATF). It performs mostly the same architectural initialization tasks as BL1 and allows to override any of the previous initializations done. Additionally, it continues the platform initialization and enables the power controller device. It

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

subsequently loads BL33, however, if a secure payload with corresponding Secure Payload Dispatcher (SPD) service is available, it also starts BL32.

2.4.4 BL32: Secure-EL1 Payload (optional)

This stage is executed on level S-EL1 and often resembles a Trusted OS (TOS). It is implemented on top of a TEE, and continues to execute in parallel to the Rich OS (ROS) and its bootloader (see BL33 and subsequent stages). On ARM, it uses their TEE implementation named TrustZone, to run an isolated and secure OS with a small resource and code footprint which only loads signed TAs. An open-source implementation exists, called OP-TEE. More on that in section 4.4.6.

2.4.5 BL33: Non-trusted Firmware

Executed on level EL1 or EL2, this stage resembles software which one may have in mind talking about traditional bootloaders. It finishes the system initialization and prepares everything for the start of the ROS, which will run in EL1. Most often, it also provides additional functionality for various support tasks. Again, an Open-Source implementation exists as U-Boot, which will be revisited in section 4.4.7. It is also described in chapter 9 of [58], where its concepts and setup to load an embedded Linux system are explained. This stage ends the standard boot process.

2.4.6 Secure Boot

To get a secure or trusted boot process now, one has to leverage trusted computing, which uses hardware and software to provide security to the system. Patterns to use this for the boot process and subsequently also secure storage are presented by Löhr et al. in [3]. They label secure boot “the heart of most security solutions” and aim at presenting common patterns to enhance OS security. An example how this can be done using TPMs is also presented in chapter 6 of [30], and Kai et al. in [59] take this to embedded systems by using an MTM together with U-Boot and Linux. In [60], Khalid et al. write about the implementation of trusted boot for embedded system. As it is beneficial to integrate the whole functionality in one SoC, they use a Field-Programmable Gate Array (FPGA) to show the usefulness of their design against software attacks. However, they also state their solution does not help against physical attacks. A full review on different secure boot implementations flow of embedded applications was conducted by Rashmi et al. in [61].

Establishing now a secure boot flow is all about building a CoT, starting with some RoT – typically, this is the initial bootloader code in the ROM of the SoC. To be able to authenticate code loaded later in time, a public key gets written to some write-once fuses. The ROM boot code uses this key to check the signature on the standard bootloader program. If the check fails, the ROM boot code rejects loading the bootloader. Otherwise, the bootloader itself checks the signature of the next software stage, the OS, extending the CoT. This way, it has to work for each step. The abstract schematics described

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

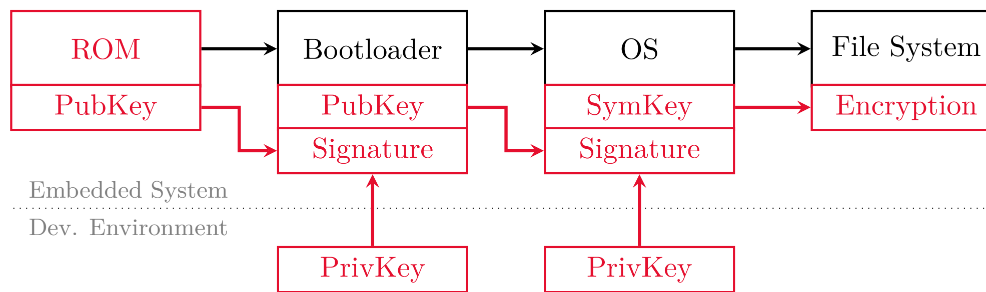

Figure 2.3: Chain of Trust example on generic secure boot flow

there can be seen in figure 2.3. Using this technique, also more boot flow stages can be used. However, including a file system poses a new challenge, as working with signatures typically does not work here. This will be outlined in the next section.

2.4.7 CoT Extension

The state of a file system is constantly changing, as its purpose is persistent data storage. Therefore, the initial file system cannot just be signed and its signature checked while loading it to ensure its authentication. To achieve that, it needs to be encrypted. Different approaches exist in this area, it is possible to either encrypt the whole device below the file system layer (e.g. using `cryptsetup`[6]) or harness per-file file system level encryption (e.g. `fscrypt`[7]). One challenge they all have in common, though: On this kind of embedded systems we are talking about, there is no one to feed the encryption key (referred to as *SymKey* in figure 2.3) to the system. For security in personal computers, secure storage is widely aided by TPMs today. However, most of embedded systems do not provide the same modules and therefore have to use other techniques to securely store a file system master key and achieve secure storage. Storing a symmetric key in an (unencrypted) OS binary blob is inherently unsafe, therefore, another possibility has to be found for key management in such environments, satisfying various security and system requirements. This can be done, for example, by using trusted computing in file system development, as Jin et al. in [62] show. However, also if the master key is stored securely, one drawback remains: All the keys still remain in system memory while in operation. To further improve this, Yu et al. in [63] briefly outlined the idea of routing all file system encryption operations through an HSM. Although done in a server environment, this may turn out (partially) applicable for smaller systems.

2.5 Threat Modeling for CPS

Nowadays, threat modeling is considered a standard tool in all development environments. As Adam Shostack put it in his famous book: "Threat modeling is the key to a focused

[6] `https://gitlab.com/cryptsetup/cryptsetup`
[7] `https://www.kernel.org/doc/html/latest/filesystems/fscrypt.html`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

defense. Without threat modeling, you can never stop playing whack-a-mole."[64]. How to do it right is a different question, though. Starting with the increased publication of attack data over the Internet and through Computer Emergency Response Teams (CERTs), Moore et al. in [65] started with creating an approach of attack modeling. Aimed on security analysts and system designers, the patterns should help them to identify common attacks. However, threat modeling matured over the years, and today we got various different approaches as also a heterogeneous information system environment, each with their own recommendations.

Threat and risk modeling in vehicular systems was examined by Kadhirvelan et al. in [66]. Highlighting the connectedness of safety and security in vehicles, they analyze different methodologies and conclude, that all of these had to be modified to be applicable to standard vehicular systems. That shows, that choosing the right methods, adapted to the use case, is quite important. Another publication which shows this step-by-step for embedded systems in the automotive scope was provided by Hadding et al. [67]. To be able to compare the different existing methods, Shevchenko et al. recently summarized available modeling techniques, also mentioning CPS which may be vulnerable to nontraditional threats [68]. Overall, they reviewed 12 different threat modeling methods (which are not necessarily comprehensive) and do not recommend a special method, as this is a decision to take based on the needs of the project where it shall be used. Consecutively, in [69] they continue with an evaluation of the different methods for systems of CPS. The threat modeling method one chooses should fit one's system and target aspects, ranging from traditional to safety-related vulnerabilities and address kinetic, physical, cyber-physical, cyber-only, supply chain and insider threats. Finally, they recommend a combination of different methods including STRIDE.

The mnemonic of Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege (STRIDE) is mentioned in the bible of threat modeling by Adam Shostack, *threat modeling: designing for security* [64]. He covers the details of using STRIDE and all other things to watch out for while modeling threats in a system. The next book by Howard et al. is about its integration into the Microsoft Security Development Lifecycle (SDL) [70]. This is a 12-step process to ensure well-crafted security throughout the whole development process. For a further risk assessment, also a mnemonic of Damage, Reproducibility, Exploitability, Affected Users, Discoverability (DREAD) is covered here. STRIDE is often referred to as the most mature threat modeling method, therefore also a lot of work has already been done in this area. One is a descriptive study on STRIDE by Scandariato et al. in [71] to quantify its cost and effectiveness. Evaluating over three years, they found their results quite satisfying, although they admit an objective measurement is quite difficult. Also other methods were created on-top of STRIDE, as the hybrid threat modeling method by Mead et al. in [72], coupling it with Security Cards and Persona non-Grata (PnG). For the domain of CPS, an approach was presented by Khan et al. in [73]. They propose a comprehensive five-step threat modeling framework built on STRIDE and Data-Flow Diagrams (DFDs) and finally state, that "STRIDE is a light-weight and effective threat modeling methodology

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

for CPS that simplifies the task for security analysts to identify vulnerabilities and plan appropriate component level security measures at the system design stage".

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Analysis

In the following sections, after explaining the used methodology, a generalized TRA will be conducted. This TRA will be used to examine, if and how available secure hardware targeted at V2X devices will be applicable to improve the security of C-ITS stations.

3.1 Frameworks

Our methodology consists in applying existing and established frameworks, as also Soja et al. especially stated the need for standards in system design and development [27]. Some basics on them will be explained in this section, and we will add and define our adaptions here. The requirements are conducted accordingly to the IEC 62443, to make a later application to the full system possible. The TRA will use STRIDE and DREAD for CPS, loosely following the method in [73].

3.1.1 System: IEC/ISA 62443

The standard IEC 62443, formerly known as ISA 99[1], is a recently developed collection of standards (some parts are still under development, though), targeting IoT and CPS systems and claiming ISO 27001/27002 compatibility for easier integration in existing environments. It is targeting product suppliers, as well as system integrators and asset owners and deals with single components as also network systems.

The basic concepts and terminology are explained in IEC 62443-1-1 [74]. It builds on defense-in-depth, "applying multiple countermeasures in a layered or stepwise manner", and also TRA recommendations including classes of threats. Taking a qualitative approach, general Security Levels (SLs) are recommended from 1 to 4. These levels are used to assess the targeted, achieved and capability SL.

[1] https://www.isa.org/isa99/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Level	Intention	Resources	Skill	Motivation	Attacker Examples
SL 1	good	○	○	○	Casual User
SL 2	bad	○	∗	○	Insider, Hacker
SL 3	bad	∗	●	∗	Hacktivist, Terrorist
SL 4	bad	●	●	●	Nation States, APTs

● High, ∗ Medium, ○ Low

Table 3.1: IEC 62443 security levels

A summary of all SLs is given in table 3.1, they are defined as follows:

- **SL 1:** Prevent the unauthorized disclosure of information via eavesdropping or casual exposure.
- **SL 2:** Prevent the unauthorized disclosure of information to an entity actively searching for it using simple means with low resources, generic skills and low motivation.
- **SL 3:** Prevent the unauthorized disclosure of information to an entity actively searching for it using sophisticated means with moderate resources, IACS specific skills and moderate motivation.
- **SL 4:** Prevent the unauthorized disclosure of information to an entity actively searching for it using sophisticated means with extended resources, IACS specific skills and high motivation.

The core of the standard are the Foundational Requirements (FRs). These seven classes are subsequently used in the proceeding standards, such as IEC 62443-4-2 [75], defining "technical security requirements for IACS components". Meant for vendors of components for bigger systems, these seven FR groups containing multiple Component Requirements (CRs) including a baseline and Requirement Enhancements (REs) (to reach the targeted SL) are defined here. Specifically targeted on hardware are the following CRs:

- CR 1.5 Authenticator Management
- CR 1.9 Strength of public key-based authentication
- CR 1.14 Strength of symmetric key-based authentication

Also all of the six given Embedded Device Requirements (EDRs) apply here.

3.1.2 Threat Analysis: STRIDE

A threat analysis typically starts with setting the scope and modeling the system, e.g. by its data flows. After this is done, the existing threats shall be found and categorized. STRIDE, already mentioned in section 2.5, is a mnemonic to assist in threat modeling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The explanation of the single letters from [64] is:

- **Spoofing** is pretending to be something or someone you're not.
- **Tampering** is modifying something you're not supposed to modify. It can include packets on the wire (or wireless), bits on disk, or the bits in memory.
- **Repudiation** means claiming you didn't do something (regardless of whether you did or not).
- **Denial of Service** are attacks designed to prevent a system from providing service, including by crashing it, making it unusably slow, or filling all its storage.
- **Information Disclosure** is about exposing information to people who are not authorized to see it.
- **Elevation of Privilege** is when a program or user is technically able to do things that they're not supposed to do.

To find mitigation solutions afterwards, it is useful to assess the risk these threats pose to the system.

3.1.3 Risk Analysis: DREAD

A risk assessment shall help with realizing the risk of different threats. However, one should keep in mind that, as hard as you may try, it "remains a statistical estimation that inherently includes uncertainties" [9]. DREAD, described in [76], is another mnemonic, helping you with estimating the risk. Its letters stand for:

- **Damage Potential:** How great is the damage if the vulnerability is exploited?
- **Reproducibility:** How easy is it to reproduce the attack?
- **Exploitability:** How easy is it to launch an attack?
- **Affected Users:** As a rough percentage, how many users are affected?
- **Discoverability:** How easy is it to find the vulnerability?

As also mentioned in a Microsoft blog post[2], DREAD got quite a bit of criticism. Therefore, we use a modified weighting approach here. First, our used rating includes four values from 0 (none) to 3 (high). Furthermore, as

$$\text{risk} = \text{damage} \times \text{probability}$$

we split DREAD into

$$\text{damage} = \frac{1}{3}(Da + R + A) \qquad \text{probability} = \frac{1}{2}(E + Di)$$

which means, that both are now values from 0 to 3 and our overall risk contains values from 0 to 9. This may be interpreted as

- $0 \leq v < 2$: low risk
- $2 \leq v < 6$: medium risk
- $6 \leq v \leq 9$: high risk

[2] https://blogs.msdn.microsoft.com/david_leblanc/2007/08/14/dreadful/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

3.2 Requirements Analysis

Although quite shallow, the rejected delegated directive [11] demands the use of a "secure hardware module" to protect secrets and enhance device security. The IEC 62443 standard is more specific here. Our targeted SL will be set to 3, covering most of the advanced attackers (see table 3.1). In the following, we give selected hardware-connected CRs and EDRs (and, if needed, their corresponding REs), which we will use as basic system requirements:

- CR 1.5 Authenticator Management
  RE (1) Hardware security for authenticators
- CR 1.9 Strength of public key-based authentication
  RE (1) Hardware security for public key-based authentication
- CR 1.14 Strength of symmetric key-based authentication
  RE (1) Hardware security for symmetric key-based authentication
- EDR 3.2 Protection from malicious code
- EDR 3.10 Support for updates
  RE (1) Update authenticity and integrity
- EDR 3.11 Physical tamper resistance and detection
  RE (1) Notification of a tampering attempt
- EDR 3.12 Provisioning product supplier roots of trust
- EDR 3.13 Provisioning asset owner roots of trust
- EDR 3.14 Integrity of the boot process
  RE (1) Authenticity of the boot process

The listed requirements above apply to the whole system, however, services running on the stations are another source of requirements, as they also often need confidentiality, integrity and availability. These will be covered in section 3.2.1.

Another requirement recently stressed by many is crypto agility. With a rapidly changing area of threats and the shadow of quantum computers (which will render today's asymmetric cryptography methods useless) above, building systems with hard-coded security features is not a good idea, and also projects in Europe have become aware of that. As Lonc et al. state, "they require capability to improve crypto-algorithms over time in C-ITS system which is a major issue in embedded systems due to constrained resources (i.e., Hardware Security Module, crypto-accelerators)" [8].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2.1 Services

In this section, typical C-ITS station services are gathered. A brief explanation is given, why and how they may profit from features provided by secure hardware.

Administration Front-End

Used mainly for convenient configuration and administrative purposes, a webserver is most often included in such devices. This means, it acts as a gate for users and should provide secure HTTP with Transport Layer Security (TLS) to provide confidentiality, integrity and privacy of the exchanged data. To provide TLS, the server possesses a root certificate (which is basically an accredited public key), signed by a certification authority, with which it now can prove its identity. The also included encryption is done by creating a symmetric session key with the help of the Diffie-Hellman (DH) key exchange. This now leads to a long-term certificate and short-term session keys which need protection.

Administration Back-End

Of course, administration is also often done through other channels. One thing they have in common is, that they are mainly also protected by adding a TLS layer, leading to the same protective needs as mentioned above. An example for this would be an API via the modern WebSocket protocol.

Another standard tool in administrating devices scattered over various locations are Virtual Private Networks (VPNs). As its name says, a VPN server creates a virtual network with all connected clients. To make the network "private" and ensure confidentiality and integrity, two main variants are common: Using Pre-Shared Keys (PSKs) or a PKI. The first variant denotes sharing a secret key which is used for authentication, the second variant works with certificates, which are created by the server. Each client gets one to prove their identity against the server, which checks if they are valid. Therefore, independent from the fact if the device takes the role of the server or the client, it either leaves us with a shared, secret key or a certificate for server/client authentication to protect.

Sensor Interfaces

Built for providing and routing data, the flexibility and extendability plays a big role for C-ITS stations. Therefore, they often provide interfaces to communicate with external sensors. However, as street- and intersection spaces can be huge, they may not be positioned directly at the station's position, so they typically use WiFi or Ethernet networks. Using best practice again, these are most likely also secured by adding a TLS layer, the protective needs of which we already covered.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

V2X Communication

As mentioned in section 2.1, the secure part of V2X communication also relies on a PKI, although now with a different authority in contrast to TLS. Again, this means we have to protect a certificate used for signing outgoing messages.

Secure Storage

Secure storage is a service more integrated in the system and was also already mentioned in section 2.4.7. It is essential in ensuring the confidentiality and privacy of system and user data. As it cannot be integrated into the secure boot flow, it needs to get an extra (symmetric) key for its encryption. This key is ideally unique on a device basis and needs to be kept secret, therefore it is another asset to protect.

Key Management

As we see from the previous services, all to some extent demand a possibility to securely manage secret keys. This is also true for some system requirements of section 3.2, as the RoT of a secure boot implementation is just a private key which also needs integrity protection. Additionally, the same is true for confidential updates. Whereas updates can be authenticated by using a public key, the system has to be protected from altering it and confidential updates demand storing a symmetric key somewhere. This key also needs a good protection, as it may not be device unique. This would result in unique updates for every device, which is not feasible in practice.

Many applications providing one of the first three services (e.g. Nginx and OpenVPN) are compatible to OpenSSL and use it for their cryptographic operations, like en- and decrypting data, creating and verifying signatures and key exchange mechanisms like DH. OpenSSL already provides an API with the intention to make it easier to leverage secure hardware to extend the security of the performed operations. There also exist standardized cryptography interfaces like PKCS#11, which make it easier to find or create compatible software. These APIs may reduce the effort in making use of secure hardware for a lot of services, as they can further rely on their OpenSSL implementation, while OpenSSL relays all operations to a secure hardware module.

3.3 Threat and Risk Analysis

For the TRA, we will follow the steps below. They are recommended by [69] and slightly adapted:

1. Define technical scope
2. Decompose system and create model
3. Identify threats
4. Rate threats and calculate risk
5. Find mitigation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

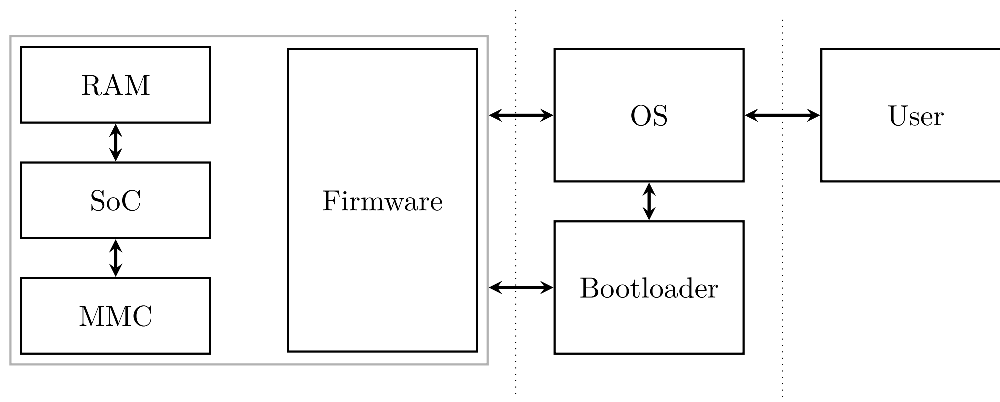

Figure 3.1: System model for TRA

3.3.1 Scope

Our chosen scope is rather small, we will focus on threats which involve hardware or low-level software, as also the requirements in section 3.2 show. Therefore, we will not cover threats posed by application software or its implementation, as also specific threats of network devices. Additionally, we do not consider supply chain attacks and physical attacks on the SoC itself, as such kind of attacks require excellent skills, high effort and many resources and hence overshoot our required security level. Subsequently, we take a look at three different types of attackers: Attackers with physical access to the device, attackers with user access and – aligned with the defense in dept approach of IEC 62443 – attackers who already got root access on the system. Our modeled system is presented in the next section.

3.3.2 System Model

> *"What are you building?"* [64]

To start with modeling, you need to know your system at the right level of abstraction. Therefore, figure 3.1 shows the system model with its data-flow used in our subsequent TRA. The dotted lines depicture trust boundaries in between the modules. The mentioned attacker types strongly correspond to one or more components of the model. Components targeted by attackers with physical access are RAM and ROM, the component targeted by unprivileged attackers is the user space and the targets of privileged attackers are the OS and the bootloader.

3.3.3 Threat and Risk Modeling

> *"What can go wrong?"* [64]

Using STRIDE and DREAD, the results of our TRA can be seen in table 3.2 and table 3.3. We can draw some interesting conclusions from that.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As high risk is defined as 6 and above, this includes:

- **Threat 07:** As the ROM is easier to exploit, similar threats (like 01 and 07) are more risky targeting the ROM. In general, the hardware components need some additional protection.
- **Threats 12-19:** Nearly all OS component threats combined with privileged attackers are extremely dangerous. A defense-in-depth approach can help us with that.
- **Threats 22, 23 and 25:** These threats show that securing updates is also very important, as they pose a great risk to the system.

Overall, the results definitely show a need for secret keys/certificate and update protection. Finally, as no threat is rated below risk 2, none of them can be considered low risk. Although the high effort for side-channel attacks lowers their risk, mitigation is strongly encouraged.

3.3.4 Mitigation

> *“What should you do about those things that can go wrong?”* [64]

After analyzing threats and their connected risk, hardware modules for threat mitigation are searched and examined. As we want to see which improvements hardware-based security features enable and which we can leverage, the next section will present some of them in a market survey.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 3.2: Threat Analysis

#	Type	Comp.	Threat	Impact
01	I	RAM	read secret keys or certificates offline	control this and other devices
02	D	RAM	destroy physically	brick system
03	T	ROM	modify FW/BL/OS offline	inject malware
04	T	ROM	modify FW/BL/OS offline	reuse hardware
05	R	ROM	modify logs	hide attack
06	I	ROM	modify FW/BL/OS offline	extract system information
07	I	ROM	read secret keys or certificates offline	control this and other devices
08	I	ROM	read FW/BL/OS offline	explore potential vulnerabilities
09	D	ROM	modify FW/BL/OS offline	brick system
10	D	ROM	destroy physically	brick system
11	E	ROM	modify FW/BL/OS offline	control this device
12	S	OS	load own BL/OS	control this device
13	T	OS	modify FW/BL/OS	inject malware
14	T	OS	modify FW/BL/OS	reuse hardware
15	R	OS	modify logs	hide attack
16	I	OS	read secret keys or certificates from ROM	control this and other devices
17	I	OS	read secret keys or certificates from RAM	control this and other devices
18	I	OS	read FW/BL/OS	explore potential vulnerabilities
19	I	OS	modify FW/BL/OS	extract system information
20	D	OS	modify FW/BL/OS	brick system
21	T	User	modify RAM via SC	control this device
22	T	User	modify BL/OS update	control this and other devices
23	T	User	use old BL/OS update	restore vulnerable software
24	I	User	read secret keys or certificates from RAM via SC	control this and other devices
25	I	User	read BL/OS update	explore potential vulnerabilities

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 3.3: Risk Analysis

#	Da	R	E	A	Di	Damage	Prob.	Risk
01	3	3	2	3	1	3.0	1.5	4.5
02	1	3	3	1	3	1.7	3.0	5.0
03	3	3	2	2	2	2.7	2.0	5.3
04	3	3	2	2	2	2.7	2.0	5.3
05	1	3	2	2	2	2.0	2.0	4.0
06	3	3	2	2	2	2.7	2.0	5.3
07	3	3	2	3	2	3.0	2.0	6.0
08	1	3	2	2	2	2.0	2.0	4.0
09	1	3	2	1	2	1.7	2.0	3.3
10	1	3	3	1	3	1.7	3.0	5.0
11	3	3	2	2	2	2.7	2.0	5.3
12	3	3	3	2	3	2.7	3.0	8.0
13	3	3	3	2	3	2.7	3.0	8.0
14	3	3	3	2	3	2.7	3.0	8.0
15	1	3	3	2	3	2.0	3.0	6.0
16	3	3	3	3	3	3.0	3.0	9.0
17	3	3	3	3	3	3.0	3.0	9.0
18	1	3	3	2	3	2.0	3.0	6.0
19	3	3	3	2	3	2.7	3.0	8.0
20	1	3	3	1	3	1.7	3.0	5.0
21	3	2	1	2	1	2.3	1.0	2.3
22	3	3	2	3	3	3.0	2.5	7.5
23	2	3	3	2	3	2.3	3.0	7.0
24	3	2	1	3	1	2.7	1.0	2.7
25	1	3	3	2	3	2.0	3.0	6.0

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.4 Market Survey

According to the requirement of using a “secure hardware module”, the following pages will give an overview of several stand-alone and SoC-integrated modules.

3.4.1 Secure Hardware Modules

NXP SXF1800

The SXF1800[3] is an HSM especially targeted at supporting standardized V2X communication as in IEEE 1609.6 and ETSI TS 103 097. Additionally, it provides long-term key storage for certificates and other data. Connected to the SoC via Serial Peripheral Interface (SPI), it provides 1MB of storage for user data. NXP claims crypto agility through secure firmware updates, supporting National Institute of Standards and Technology (NIST) and Brainpool Elliptic Curve Cryptography (ECC) curves. It is CC EAL5+ certified, compliant with the Car-to-Car (C2C) V2X HSM protection profile and the Federal Institute Processing Standards (FIPS) 140-2 level 3 requirements.

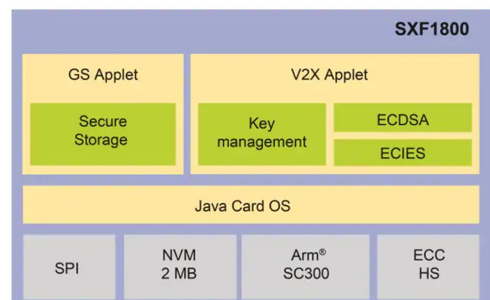

Figure 3.2: NXP SXF1800 HSM block diagram (www.nxp.com)

[3]https://www.nxp.com/products/wireless/dsrc-safety-modem/secure-element-ic-for-v2x-communication:SXF1800

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Infineon Optiga TPM SLI 9670

The Optiga TPM SLI 9670[4] is a TPM targeted on industrial and automotive applications and provides key store and management functionality. An overview is given in figure 3.3. The connection to the SoC is realized with SPI. Again, crypto agility through updateability is a topic here and the related Linux drivers are open-source. The currently supported cryptographic algorithms include HMAC, SHA1, SHA2, ECC (BN-256, P-256), AES and RSA [77]. The module conforms to the TCG standard TPM 2.0 and is certified according to CC EAL4+. Evaluation modules are available together with documentation on how to run these together with Linux on a Raspberry Pi 3 [78] and 4 [79].

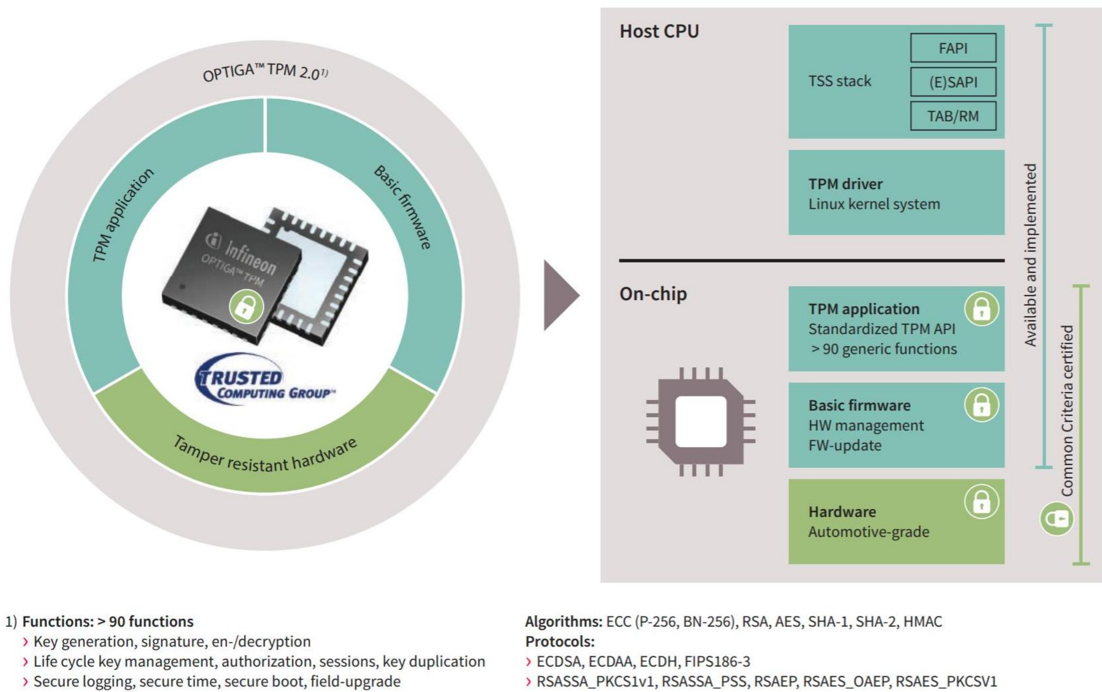

Figure 3.3: Infineon OPTIGA TPM software and features [80]

[4] https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/sli-9670/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.4.2 SoCs

Autotalks Craton 2

The Craton 2 platform[5] was specifically designed for connected vehicles. Created by the Israeli company Autotalks, this resembles a product not built from one of the two big manufacturers in this market and marketed as a cost-optimized solution. It combines a 32-bit dual-core ARM Cortex A7 with a WiFi modem supporting 802.11p, a Cellular V2X (C-V2X) modem and a specialized V2X HSM. This HSM supports ECDSA and various hardware accelerators. They claim to be the first and only company supporting both competing V2X technologies in a single SoC. Pre-integrated software is available. An evaluation kit is ready to purchase, however, overall not a lot of information could be found on the SoC.

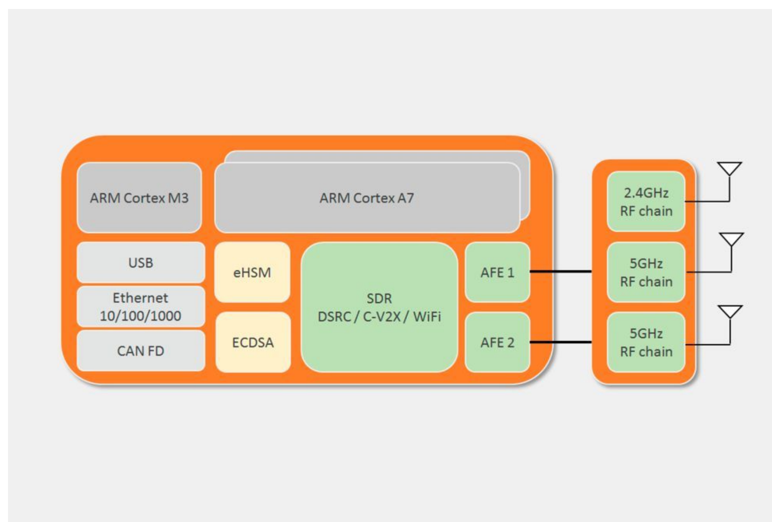

Figure 3.4: Autotalks Craton 2 block diagram (www.elektronikpraxis.vogel.de)

[5] https://www.auto-talks.com/product/craton2/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Infineon AURIX TriCore TC3xx

The AURIX TriCore TC3xx[6] SoC family is the most recent and powerful microcontroller product line of Infineon. Consisting of numerous different types containing up to 6 cores, their "TriCores" are 32-bit RISC cores running at about 300MHz. The documentation [81] includes an explanation of their security features and a lot more information. On the hardware side, an HSM is embedded, which can be used for storing cryptographic keys and also includes dedicated hardware accelerators (supporting AES128, ECC256 and SHA2). They also claim crypto agility for their HSM implementation, as it is programmable via software. Due to the flexibility of their HSM, trusted customized vendor apps are possible, which are executed in a TEE on the HSM. Additionally, a partly open-source software stack including a complete toolchain is provided by them. The embedded HSM fulfills the full EVITA standard. Various "Triboards", their evaluation kits, are available to support the development.

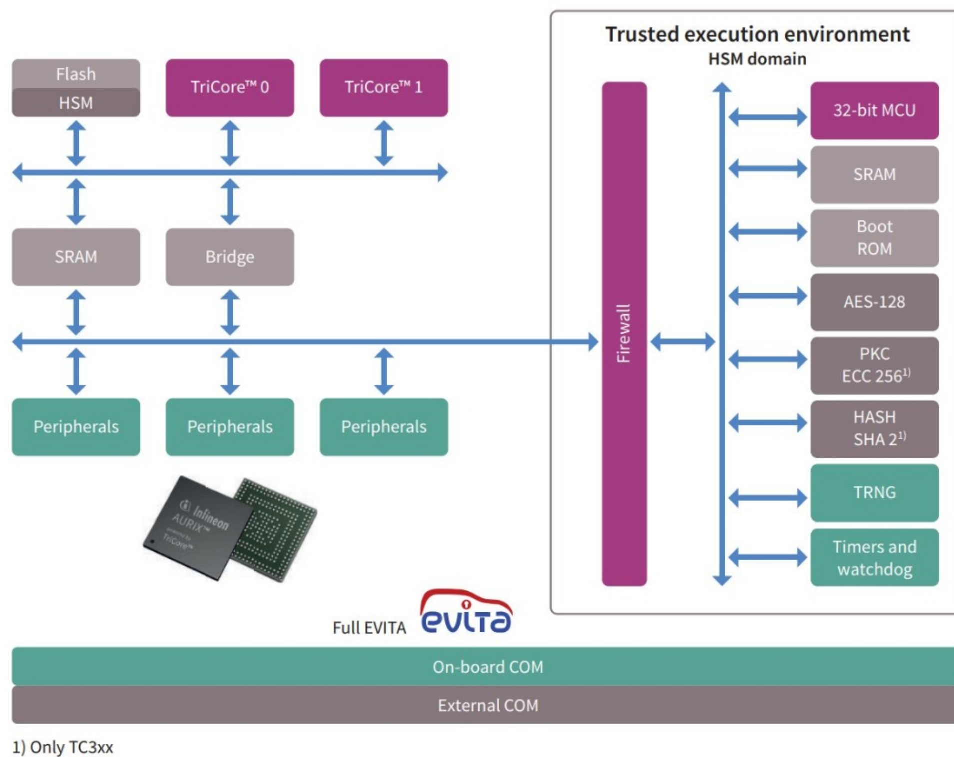

Figure 3.5: Infineon Auric TriCore security diagram (www.infineon.com)

[6] https://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-microcontroller/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

NXP i.MX6

The i.MX6[7] SoC family is well-established and widely used in industry and automotive products. It consists of 9 different variants, built around up to four 32-bit ARM Cortex A7 or A9 cores, running at a speed of about 1GHz. An NXP implementation for secure boot is available, called High Assurance Boot (HAB)v4, as also a cryptographic cipher engine module called Cryptographic Assertion and Assurance Module (CAAM). A lot more information can be found in its reference manual [82]. The current software stack is extensive and mainly open-source [83]. It includes U-Boot, OP-TEE and Linux Kernel version 4.14. Various evaluation kits are available for testing.

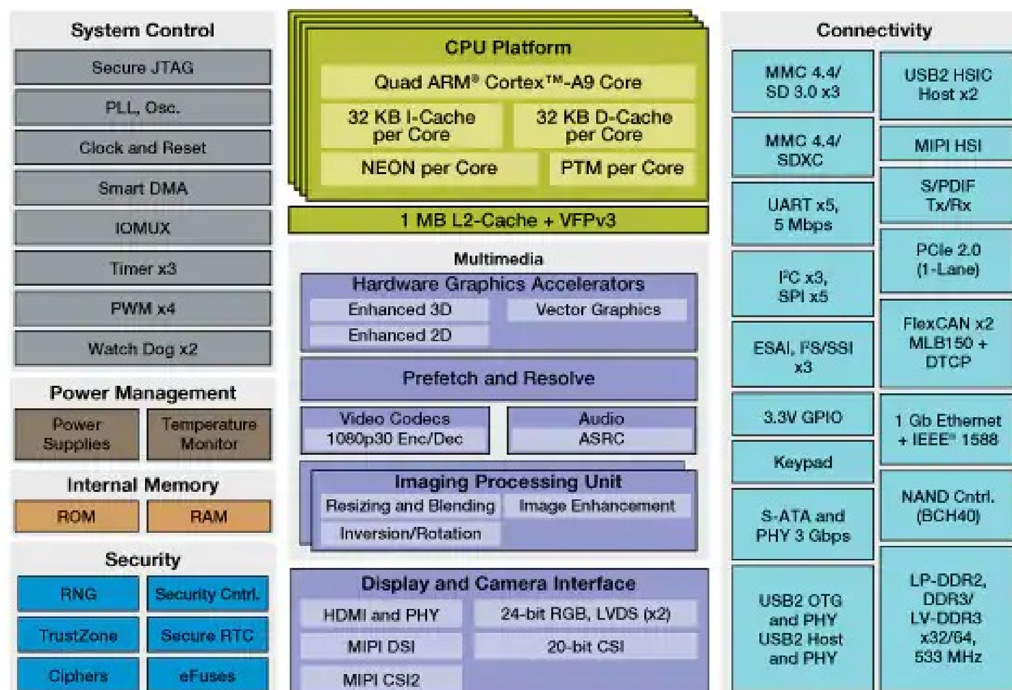

Figure 3.6: NXP i.MX8 SoC features (www.nxp.com)

[7] https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i.mx-applications-processors/i.mx-6-processors:IMX6X_SERIES

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

NXP i.MX8

The i.MX8[8] SoC family is the most recent product line of NXP. It currently consists of 5 SoCs, containing up to four 64-bit ARM Cortex A53 or A35 cores and an additional M4 co-processor. The firmware and security features are now handled by two other dedicated co-processors not available for programming. The secure boot implementation was improved, now called Advanced High Assurance Boot (AHAB), and a Secure Hardware Extension (SHE) added to the CAAM, now supporting AES, 3DES, RSA, SHA1, SHA2 and MD5. It also works as secure key storage and the inline encryption engine speeds up AES128. Additionally, it provides 10 dedicated active and passive tamper-protection pins for detecting tampering attempts. Again, a lot more information can be found in the reference manual [84] and the security reference manual [85]. All functionality is summarized in figure 3.7. As in the i.MX6 series, the current software stack is extensive and mainly open-source [83]. It includes U-Boot, OP-TEE and Linux Kernel version 4.14. Various evaluation kits are also available for testing.

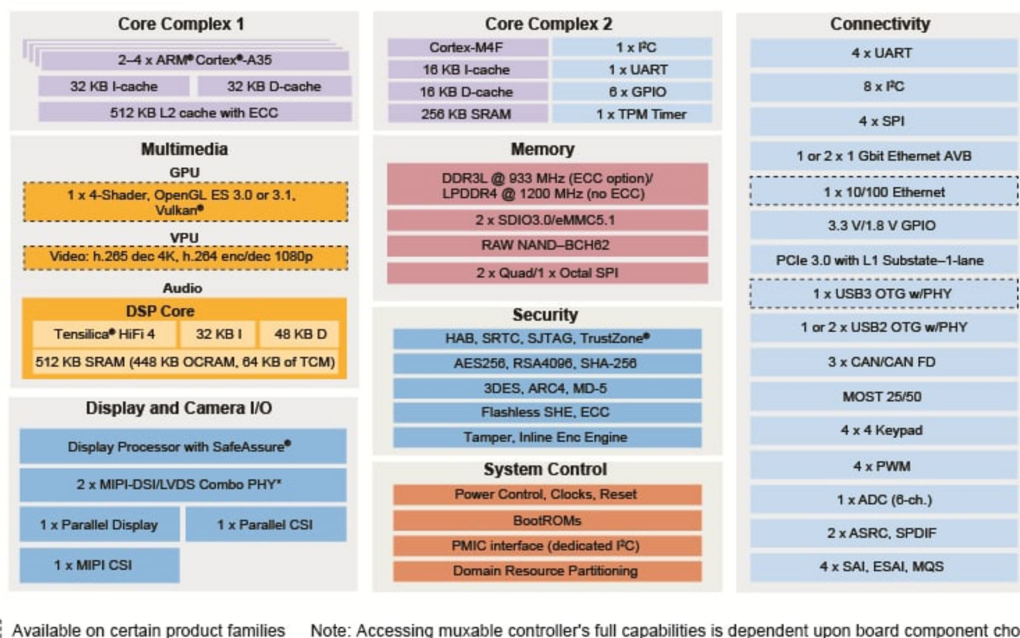

Figure 3.7: NXP i.MX8 SoC features (www.nxp.com)

[8] https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i.mx-applications-processors/i.mx-8-processors:IMX8-SERIES

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.5 Evaluation

On our selection of one HSM, one TPM and four SoCs, the survey shows a lot of similarities as also a lot of differences. The most basic differences lie in the nature of HSM and TPM modules. The main task of the first is to support cryptographic operations and store cryptographic keys, whereas the focus of TPMs also includes the support of secure boot by providing a RoT and detecting tampering attempts. However, the boundaries get more and more blurred here. SoCs often include a mixture of different features, not necessarily bundled in a single module. For a full evaluation, a lot more work has to be done on each product, however, due to the limited scope and the huge extent of the connected topics, this is only possible for one product in this work. The short survey of section 3.4 shall now help to choose one product worth of further examination.

3.5.1 Available Information

The main problem we ran into here was the (not) freely available information from the companies. In this market, it strongly depends on the brand, how much information is given to you on the products without having to sign a Non-Disclosure Agreement (NDA), which makes it impossible to publish the work. This is also true for their code – there is a difference, if something is available as open-source Git repository or as "open-source" code archive you get after signing an NDA. Therefore, we were strongly limited in our short analysis by this fact.

3.5.2 Requirements

The developed requirements in section 3.2 show, that support for symmetric and asymmetric cryptography is important. As a core functionality of HSMs and TPMs, this is supported by all surveyed HSM and TPM modules, as well as the SoCs. In the Autotalks and Infineon SoCs, an HSM is integrated, and NXP has implemented a hybrid approach with its CAAM module. Therefore, they all support storing symmetric and asymmetric keys securely and provide cryptographic functions.

The flexibility and **Crypto Agility** is another important point. Here, we see a more differentiated field. Both vendors of the dedicated HSM and TPM note, that their products are flexible due to their software implementation. However, if these implementations are not open-source, this still leaves us dependent on the vendor for features and fixes. On the side of integrated modules, this is the same. Autotalks does not give any information about the flexibility of their HSM implementation, but Infineon talks about possible custom, secure applications running in the integrated HSM. And for the NXP SoCs there is to say, that the CAAM seems fully programmable by the user.

Another feature boosting flexibility, which is connected to the different cores the SoCs are using, are TEEs. For the ARM cores of the Autotalks and NXP chips this means, they support the standardized ARM TrustZone. Also the RISC core Infineon is using contains a TEE, however, it seems to be a non-standard implementation in this case.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

However, these TEEs are highly flexible and can be used for running custom, secure applications. And even better, if they are standardized, the secure applications can be reused with minimal effort in case of moving to newer hardware.

For the identified services, especially **Administration & Sensor Interface** services, a compatible interface to OpenSSL turned out to save a lot of hassle. Unfortunately, we could not spot any native support for OpenSSL or some PKCS#11 interface. Apparently, if needed, this has to be implemented on our own. For the **V2X Communication**, signing functionality is important. As mentioned before, due to the fact that every hardware integrates either a HSM, TPM or some hybrid module, all support RSA and ECC keys and signatures. **Secure Storage** is a different topic now. As all modules support AES cryptography, an encrypted file system can be set up and secured. However, this only helps little if no secure boot mechanism is enabled. Otherwise, attackers could simply run their own software and use the hardware module to decrypt the file system. Therefore, a secure boot mechanism is important here. This requires some RoT provided by a secure module and turned out to be more difficult information to determine. Only NXP explicitly states its support, with various versions of HAB. The **Key Management** service is again already covered with the features in the beginning of this section. All modules support the storing and using of e.g. AES keys, which can be used to implement confidential updates.

From a security point of view, the quality of these implementations can only be guessed for now, often there is little known about their (potential) shortcomings. An exception was the broken i.MX6 secure boot implementation[9] exploiting two buffer overflows in the boot ROM (CVE-2017-7932, CVE-2017-7936). However, this should be fixed in current revisions.

3.5.3 System Features

In terms of **Tampering Protection** also huge differences exist. Except some special features, these exist due to the fact that both dedicated modules are connected to the SoC via SPI. This means, they have a bigger attack surface, as attackers may eavesdrop and/or modify the communication. Here of course, SoC integrated solutions have an advantage, as this is barely possible with connections inside the chip. In general, there is not much information found about tampering protection features. Only the NXP i.MX8 series state, that they have dedicated tampering detection pins.

For **Certifications**, there seems to be an interesting distribution over our surveyed hardware. Both dedicated modules, the NXP HSM and the Infineon TPM are CC certified, the first also FIPS. The integrated modules on the other hand have no certification, however, Infineon claims its HSM EVITA compliant.

All vendors provide different variants of evaluation kits for testing their products.

[9]https://blog.quarkslab.com/vulnerabilities-in-high-assurance-boot-of-nxp-imx-microprocessors.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.5.4 Software State

Even good hardware is unusable for product integration if it does not provide a suitable software stack. Again, the biggest gap seems to be between the dedicated and integrated modules. The internal software used in the NXP HSM and Infineon TPM seem to be closed down and not available for custom modifications. The existing certifications may also play a role here. However, at least the Linux drivers seem to be open-source. NXP seems to be outstanding here, they build their software stack upon freely available forks of established open-source projects, like U-Boot and OP-TEE. Of course, this is also possible because of the ARM architecture. The Infineon SoC with RISC architecture on the other hand does not provide a lot of information about the supported software stack. This is also true for the Autotalks SoC (although its ARM architecture).

3.5.5 Decision for Concept Phase

In the end, the decision was made to use an NXP i.MX8QuadXPlus processor for implementing a PoC and further examining its security features. It provides the advantage of being a SoC, sparing the effort of choosing another evaluation platform and giving the possibility to implement a holistic security concept using secure boot. Although currently still in pre-production, it resembles a state-of-the-art SoC with a high number of (security relevant) features and – at a first glance – a well documented and open-source software stack.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Concept and Implementation

For the following concept on system authentication and key management and its implementation on an NXP i.MX8 application processor, the official evaluation kit (the NXP i.MX8QuadXPlus Multisensory Enablement Kit (MEK)) was used. A top-down picture of this board is shown in figure 4.1. The basic setup and first run is described in its *Hardware User's Guide* [86]. On the software side, NXP provides firmware and a basis of various open-source projects with support for their CPUs on CodeAurora[1]. When doing this implementation, their latest officially supported release tagged `imx_4.14.98_2.0.0_ga` was used, the latest unofficial release was `imx_4.19.35_1.1.0`. All that was done on an Ubuntu 18.04 LTS machine with Linux Kernel `4.15.0` and GCC in version `7.4.0`. The most recent toolchains for cross-compiling at the implementation time were GCC ARM AArch64 `8.3-2019.03`, GCC ARM EABIHF `8.3-2019.03` and GCC ARM EABI `8-2019q3`.

4.1 Concept Requirements

The here created concept and implementation shall enable secure key management for the system, to be able to use encrypted updates and storage. To achieve that, the first part is to use the existing RoT in the SoC to create a CoT up to the bootloader by setting up the NXP boot authentication. Though nothing new and provided by NXP, our work aims at gathering information scattered over numerous manuals, therefore building knowledge about the quite complex authenticated boot process and giving guidance for the implementation and identifying potential pitfalls. The second part is the extension of the CoT beyond the standard secure boot to the OS and its usage by other services. The concept shall provide a way to authenticate all the software up to the OS, as also to securely handle all secrets in the system which are required to apply encrypted updates

[1] `https://source.codeaurora.org/external/imx/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub

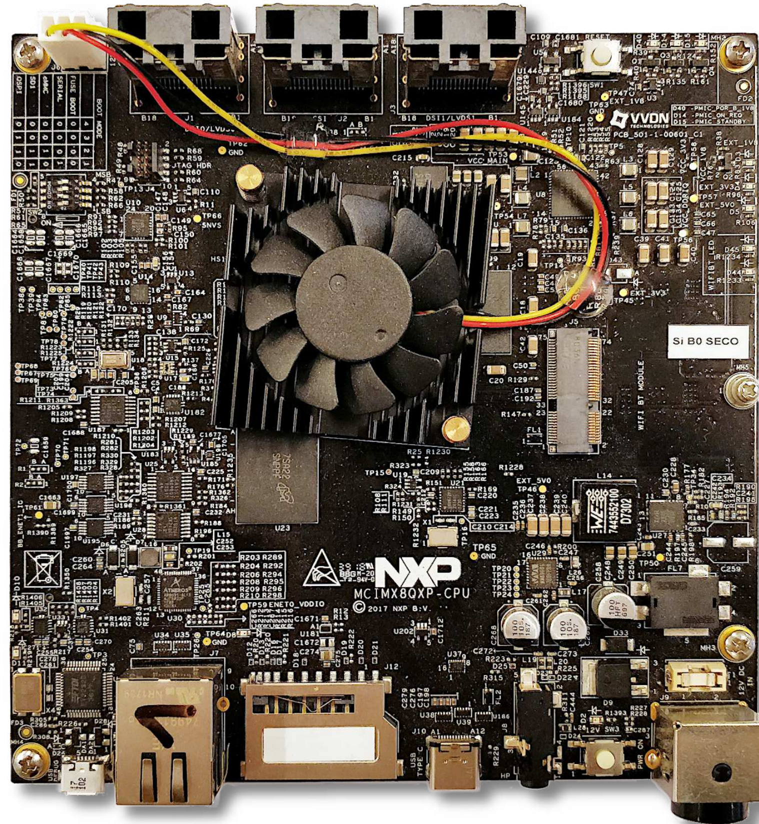

Figure 4.1: NXP i.MX8QuadXPlus MEK

and use encrypted file systems. Additionally, it shall be flexible enough to be used in further user-space applications, like OpenSSL and others from section 3.2.

4.2 Key Management Approaches

To create the concept, which is described and implemented in this chapter, we evaluated several different approaches of key-handling to ensure a sufficient security level fitting the requirements in section 3.2 to protect secret keys and certificates on multiple layers. The approaches include:

1. **FLASH:** Store key(s) in flash.
2. **FUSES**: Store key(s) in fuses.
3. **CAAM:** Use the NXP CAAM to store key(s).
4. **HSM:** Use a dedicated HSM to store key(s).
5. **TRUSTZONE:** Use the TrustZone to store key(s).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The first two are quite naïve, whereas the other three leverage special hardware features. They are examined in the following sections, an overview of the outcome can be seen in table 4.1. The chosen measures for the evaluation are three security-related and three cost-related, as in engineering the cost-benefit ratio is extremely important.

- **Remote Security:** The first measure is the security level against an attacker having no physical access to the system. This mostly targets the usual software attacks and logical side-channels, like timing measurements (e.g. RowHammer). It shall provide a measure on how good the system is shielded from malicious software
- **Local Security:** The second measure is the security level against an attacker with physical access to the system. It targets hardware attacks, like Cold-Boot or Direct Memory Access (DMA) attacks and considers the physical anti-tampering measurements as also various hardware side-channels, e.g. power side-channels or probing signals.
- **Crypto Agility:** The third security measure is the agility of the approach. As already mentioned in the requirements in section 3.2, the possibility of changing cryptographic algorithms is considered more and more important due to potential big changes in the information security domain and guarantees future-proof solutions.
- **Portability:** The first cost factor is the portability of the implementation. Can it be reused if the SoC is changed for newer hardware generations, because some application demands more performance or newer hardware revisions are available? If it cannot be reused, this requires a lot more new development effort and therefore money. Generally speaking, software tends to be more flexible here.
- **Complexity:** The second cost factor is the complexity the system requires to be programmed. The more complex the system is, the more hours one has to invest to get a sound application. Readily available driver software can lower these costs.
- **Extra Costs:** The third and last measure are the extra costs needed in addition to the SoC. Can everything be implemented with included features, or does one have to add extra hardware?

4.2.1 Approach 1: FLASH

The first, naïve approach an ingenious engineer may implement is storing a master-key in the flash and using it do decrypt local blobs of data to store more secrets. As can be easily seen, this is inherently insecure in terms of **Remote/Local Security**, as all code with sufficient privilege can read and modify the secret key and it can be physically extracted easily by directly reading the flash content, breaching confidentiality. Even as its **Agility** is good, it's a platform-independent approach in terms of **Portability** with low **Complexity** and no additional **Costs** cannot make up for it failing in the core requirements.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Approach	Security Remote	Security Local	Agility	Portability	Complexity	Costs
1: FLASH	○	○	•	•	•	•
2: FUSES	○	○	•	•	•	•
3: CAAM	•	•	∗	○	∗	•
4: HSM	•	•	∗	∗	○	○
5: TZ	•	∗	•	•	∗	•

• Good, ∗ Medium, ○ Bad

Table 4.1: Comparison of key-handling approaches

4.2.2 Approach 2: FUSES

The second, naïve approach as an improvement to the first would be not storing the master-key directly in flash, but in the SoC fuses. Fuses are used to provide One-Time Programmable (OTP) memory and are accessible like normal memory, with the write-once implication. However, in terms of **Remote/Local Security** this does not help much. As before, all code with sufficient privilege can read (though not anymore modify) the secret key as also it can again be physically extracted by directly reading the fuse memory content. However, this may be more difficult, as an in-place bootloader authentication will hinder the attacker to run their own code on the SoC and extraction from integrated SoC memory takes a lot of effort. Therefore again, even as its **Agility** is good, it's a platform-independent approach in terms of **Portability** with low **Complexity** and no additional **Costs** cannot make up its failing in the core requirements.

4.2.3 Approach 3: CAAM

The first proper approach is to leverage the CAAM provided by the SoC. As mentioned in section 3.4.2, this cryptographic module was designed to serve as RoT and provide cryptographic services. A secret key is intended to never leave the module, instead it is decrypted from a blob and saved in a special register, where it now can be used to perform cryptographic operations on data. For the **Remote Security** topic, this now leads to a quite high level. Due to the fact that the secrets never leave the dedicated module, software, independent of its privilege level, can never directly access the plaintext key. The cryptographic operations are performed within the dedicated memory of the module, which leads to the mitigation of side-channel attacks targeting the memory of the system, like RowHammer. On the **Local Security** side, the side channel resistance also stands strong. As the key never resides in the system memory, attacks like the Cold-Boot attack or DMA attacks never have any chance. Additionally, due to its integration into the SoC, communication channel probing is nearly unfeasible, as there are no accessible signals. A small downside of the CAAM, which it shares with basically all secure hardware modules, is its closed architecture and a limited possibility of updating to fix possible bugs found in the future. This also influences the **Agility** and **Portability** of this approach. Changing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

cryptographic algorithms may not be that easy, and as this module is developed and used solely by NXP and differs from chip family to chip family, the portability is quite bad here, creating a potential chip vendor dependency. This dependency is also true for the **Complexity**. The documentation of the module contains hundreds of pages, and rolling your own implementation on this level needs quite some effort. However, NXP provides a driver for its usage in Linux and a basic U-Boot driver. Finally, the **Extra Costs** are another upside of this approach, as the numerous chips of the NXP i.MX8 are shipped with integrated CAAM.

4.2.4 Approach 4: HSM

The next approach is to use a dedicated HSM, e.g. one as seen in section 3.4.1. Using a physically independent piece of hardware brings some benefits as also some drawbacks. Quite similar to approach 4, it shares a lot of its **Remote/Local Security** properties. However, a small difference exists. As it is an extra module, one has to be aware of side-channels on the connection from SoC to HSM and make sure by using its anti-tampering features, that the module cannot simply be switched to and reused on other hardware. **Agility** is an often claimed term for such modules, however, due to their closed nature it is hard to check and limited by the hardware/software boundaries. Compared to the CAAM, the **Portability** is better at least in terms of SoC independence. When switching to a newer SoC in future, one may reuse the HSM. But still, one is tied to the HSM manufacturer for hardware and new software versions, which is again important because of the **Complexity**. As such modules are quite complex, one has to rely on the vendor to provide suited drivers. Additionally to add on the downside, using an extra module means also a lot of **Extra Costs**.

4.2.5 Approach 5: TRUSTZONE

Our last approach is to use the ARM TrustZone to implement a software version of an HSM, a Soft-Hardware Security Module (sHSM). Again, this shares a lot of its properties with the preceding two approaches. A secret key never leaves the – now virtual – “module”, resulting in shielding it from software access in every aspect. This is important for **Remote Security** and typical software problems. However, there is a higher possibility for finding side-channel attacks, as, differently to the preceding two approaches, everything shares the same physical memory now. To improve the **Local Security** and shield against Cold-Boot and DMA attacks, where available, software running in the TrustZone can also leverage the on-chip RAM. Additionally, as the TrustZone is also integrated in the SoC, there are no extra communication signals which need special attention, and due to its software implementation and subsequential flexibility, one can also implement complex tasks. This flexibility is also great for its **Agility** and **Portability**: Due to its software implementation, switching cryptographic algorithms works well. The TrustZone is available in numerous ARM processors and well standardized, also making changing the SoCs easy. And although the **Complexity** of programming a software on the bare-metal TrustZone, there exist open TOSs like

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

OP-TEE, which abstracts and takes out a lot of work for the programmer. Finally, due to the integration in the ARM core, there are no **Extra Costs** to handle.

4.3 Full System Concept

As the naïve approaches 1 and 2 are unfeasible by design, they do not bring any significant improvements for our requirements. For approach 4 there is to say, that it promises the highest security, but at the cost of having to add an extra module to the system (increasing manufacturing costs) and the need of vendor support for that, building more external dependencies. Approach 3 basically has the same shortcomings, however, due to its SoC integration it is free of additional costs. We had a short look into that, however, the CAAM U-Boot driver kept crashing in all of our tests (see section 5.5). Although the used solutions in the other approaches claim a certain amount of crypto agility, approach 5 is by far the most superior here due to its entire software implementation. Additionally, with OP-TEE an already established and fully open-sourced platform exists. In the end, due to the great flexibility and portability (it's supported by all newer ARM cores) for a small security trade-off against the approaches 3 and 4, we chose approach 5 for our final concept.

In [87], people from Microsoft describe the implementation of a full-blown TPM 2.0 in the ARM TrustZone, called firmware Trusted Platform Module (fTPM). Our concept of an sHSM shall be built on OP-TEE, which drives down the implementation effort and makes it easily customizable. Compared to the full TPM 2.0 specification, the intended functionality is also way smaller. It shall provide a way for U-Boot as also for the Linux OS to securely store keys to decrypt system updates and the possibility for the latter to make use of it for other services, e.g. file system encryption. Also, applications running in Linux shall be able to leverage the sHSM for arbitrary data storage. From different sides, OP-TEE has already seen effort to create a similar service, natively integrated in OP-TEE, e.g. with a recent pull request[2] laying the foundation for secure key services. This is currently also a topic at Linaro Connect, with presentations HKG18-402[3] and SAN19-413[4].

The functional requirements for our sHSM TA deducted from the service requirements in section 3.2.1 can be summarized as

- Arbitrary secure data storage
- Secure key storage
- AES key generation and data en-/decryption
- Usable in U-Boot and Linux (for updates and system authentication)

[2] https://github.com/OP-TEE/optee_os/pull/2732
[3] http://connect.linaro.org.s3.amazonaws.com/hkg18/presentations/hkg18-402.pdf
[4] https://static.linaro.org/connect/san19/presentations/san19-413.pdf

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

with possible extension to

- RSA key generation and data en-/decryption
- Signature creation/verification
- OpenSSL interface for administration and sensor services
- ROS application access permissions

To integrate the sHSM for key-management feasibly into the system, also a trusted boot process must be set up. This includes leveraging the RoT and using the provided secure boot implementation to create a CoT and extend it to the sHSM. The sHSM now will serve as a flexible anchor for further extension to other services. This brings us to the full system concept.

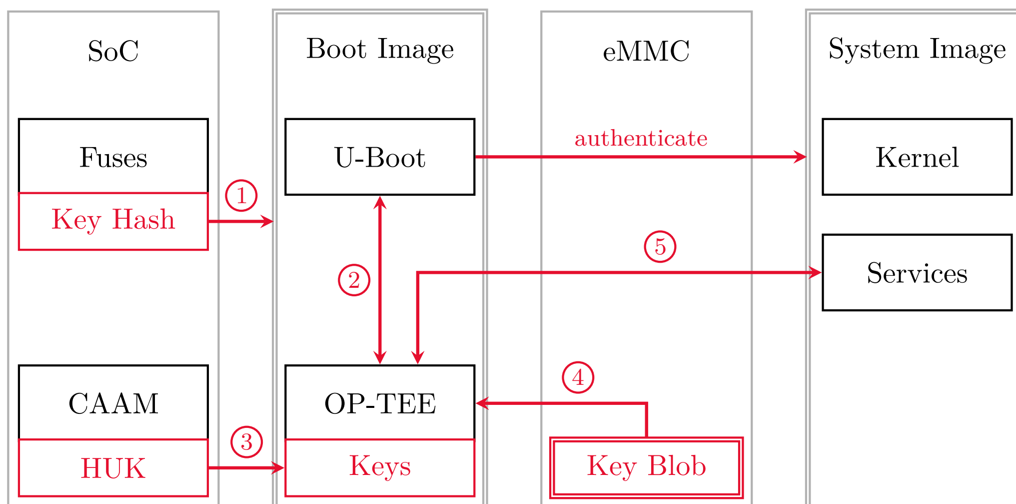

Figure 4.2: System authentication and secure services concept

A picture of the full system authentication concept can be found in figure 4.2. Cryptographic material is colored red, a doubled line means it is encrypted. Normal parts are colored black or gray, here a doubled line means the data is signed. The (1) denotes the bootloader authentication by a provided implementation from NXP (more on that in section 4.4). With the stored hash of a public key table, the whole boot image is authenticated. The next task, the kernel image authentication by U-Boot can now easily be done by signing and checking the system image. Updates can now be decrypted by U-Boot feeding the image to the OP-TEE TA (2), which uses the Hardware Unique Key (HUK) protected by the CAAM module (3) to load and decrypt the stored key blob from the eMMC (4). Then, the decrypted key is used to decrypt the image. However, not only U-Boot can use the sHSM TA now, also various services and of course the OS itself can access it in the same way (5).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.4 NXP iMX8 Boot & OP-TEE Setup

As it turned out, though provided, boot authentication can be quite complex in a modern SoC like the i.MX8. Some background on the general ARMv8 boot concept was already given in section 2.4.6. The mentioned bootloader levels are somewhat similar. However, potentially using four different processors and numerous software parts, the following sections will explain how that all fits together, before section 4.5 will explain how the NXP AHAB process works to ensure system authentication. Some of the following information was gathered and processed from [88, 84, 85] and various documentation in the official NXP sources for U-Boot [89, 90, 91]. They shall be consulted for deeper knowledge about the boot process and the boot-image format. A simplified high-level overview of the system's boot flow is depictured in figure 4.3. The M4 image, available to support the system for real-time applications, is only mentioned for the sake of completeness and is not in the scope of this work.

4.4.1 Boot Requirements

To boot the i.MX8QXP SoC, a boot-image needs to be assembled from various parts. All required files needed to build a working boot image container including OP-TEE are listed in table 4.2. The correct firmware versions for each release can be found in the *Linux Release Notes* [83]. In the next sections, these dependencies will be explained and some guidance given on how to build and integrate them.

4.4.2 BL0: ROM Bootloader

The first code that runs after a reset is the boot code in the internal ROM. It is vendor supplied and not changeable. The used SoC has two different ROMs, one belonging to a dedicated Cortex M4 processor working as System Controller Unit (SCU) and another one functioning as RoT and belonging to a dedicated Cortex M0 processor, the Security

Dependency	Filename[1]	Source[2]
ATF	`bl31.bin`	Git `imx-atf`
SECO FW	`ahab-container.img`	`firmware-imx-8.1.bin`[3]
SCFW	`imx-scfw.img`	`imx-sc-firmware-1.2.bin`
OP-TEE OS	`tee.bin`	Git `imx-optee-os`
U-Boot	`u-boot.bin`	Git `uboot-imx8`
U-Boot SPL	`u-boot-spl.bin`	Git `uboot-imx8`
Build Tool	`mkimage_imx8`	Git `imx-mkimage`

[1] Filename according to the name required by the build tool
[2] NXP Git sources from `https://source.codeaurora.org/external/imx`
[3] Closed-source

Table 4.2: Boot-image dependencies

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

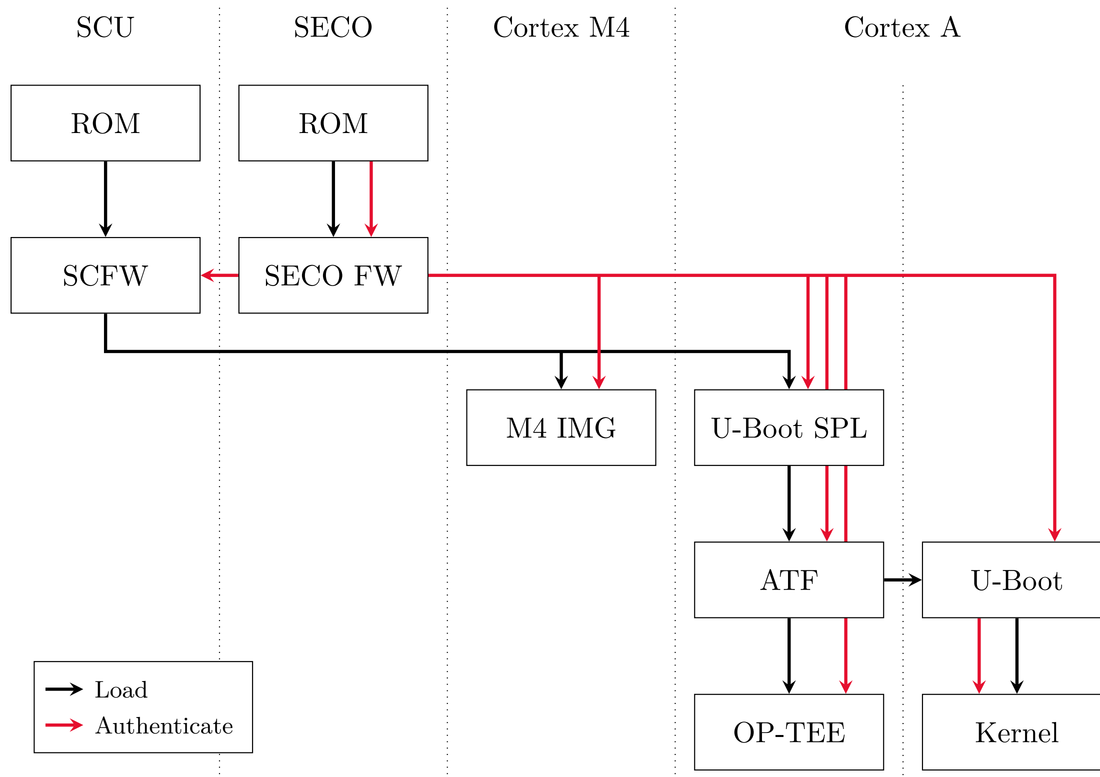

Figure 4.3: NXP i.MX8QXP boot flow and authentication

Controller Firmware (SECO). The ROM boot-code manages the boot modes and devices and loads and executes the initial firmware from the image.

4.4.3 BL1: SCFW and SECO FW

The System Controller Firmware (SCFW) is running on the SCU in the internal RAM of the SoC. It is vendor-supplied and handles resource allocation, power management and clock control. For the MEK, it is provided by NXP, for other boards it may have to be adapted (see section 5.4). An introduction to all of these can be found in a presentation on the NXP forums[5]. No repository is available for this firmware, the source code can only be obtained through the porting kit[6]. To get the pre-compiled binary, we have to download an executable binary firmware archive[7] and extract the `mx8qx-mek-scfw-tcm.bin` file.

The SECO firmware runs on the SECO, also in the internal RAM of the SoC. It handles the whole security subsystem and provides low-level security services like binary

[5] https://community.nxp.com/docs/DOC-341871

[6] https://www.nxp.com/design/i.mx-developer-resources/i.mx-software-and-development-tool:IMX-SW

[7] https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.1.bin

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

authentication. Vendor supplied and closed-source, it is signed and distributed by NXP. Again, it is contained in an executable archive[8], named `mx8qx-ahab-container.bin`.

4.4.4 BL2: U-Boot SPL

The bootloader is the first customer-supplied piece of software to start. However, this stage resembles a special bootloader, the so-called SPL. It is only needed if a secure payload (BL31, section 4.4.6) is used and was created as a small, first-stage bootloader to later load the real, second-stage bootloader (BL33, section 4.4.7). Another task it is responsible for is the insertion of the OP-TEE device node in the loaded device tree if a TEE device is physically supported. More on that in section 4.4.6. It is built together with the full U-Boot in section 4.4.7 when using the configuration file `imx8qxp_mek_spl_defconfig`.

4.4.5 BL31: ATF

Another customer-supplied firmware, tied to the ARMv8 architecture, is the ATF. It implements the ARM secure world and various ARM interface standards for the power states, trusted boot, calling conventions, control and management and software exceptions and is also able to override parameters set in the firmware stage. A lot of additional information can be found in its documentation [92, 93]. Although not vendor-supplied in general, there exists a reference implementation from ARM and NXP provides an adapted version supporting the i.MX8QXP[9]. To build it, use

```
$ make CROSS_COMPILE=aarch64-linux-gnu- PLAT=imx8qx SPD=opteed bl31
```

The `SPD=opteed` enables the built-in support in form of a secure payload dispatcher for starting OP-TEE.

4.4.6 BL32: OP-TEE

Integrated into the boot image as a special payload is also the TEE image, here OP-TEE. It includes a tiny OS which executes in the secure world, as also various supporting software. Its architecture is described thoroughly in the official documentation [43], which also contains important information about the implementation of trusted storage, secure boot, TAs and porting guidelines. To the TAs, it provides the GP interface in version `v1.1.2` [41]. The OP-TEE version adapted for the latest NXP release is `v3.2`.

An overview of the architecture can be seen in figure 4.4. It consists of four different parts, the following sections will cover all of them:

- The OS (*OP-TEE Trusted OS*)
- The kernel driver (*OP-TEE driver*)
- The helper process (*tee-supplicant*)
- The TA (*Dynamic/Static Trusted App*)

[8] `https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/imx-sc-firmware-1.2.bin`
[9] `https://source.codeaurora.org/external/imx/imx-atf/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

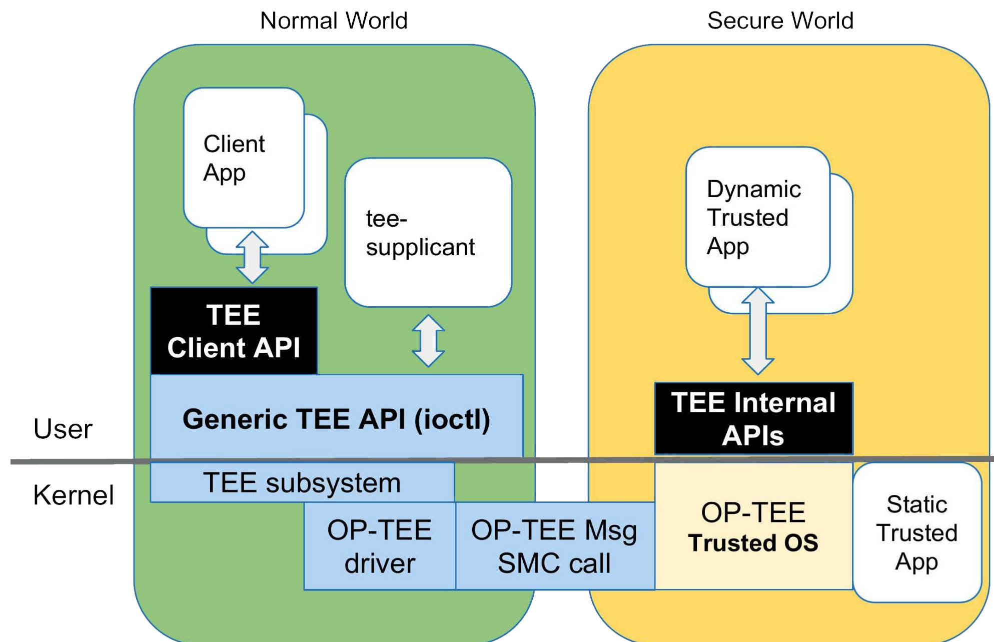

Figure 4.4: OP-TEE architecture overview (`www.linaro.org`)

Using a service provided by a TA now works in the following way: The *Client App* uses the *TEE Client API* to request the loading of a TA, which the *OP-TEE driver* relays to the *OP-TEE Trusted OS*. This *Trusted App* is typically stored signed on the Rich Execution Environment (REE) (although there are different options, see section 4.4.6) and gets loaded by the *OP-TEE Trusted OS* using the help of *tee-supplicant*. After a successful signature check, the request returns and the *Client App* can now use the full functionality of the *Trusted App* until it requests to close it. This functionality is implemented using the *TEE Internal APIs* and most of it will be solely served by the *OP-TEE Trusted OS*. However, for things like trusted storage, the help of the *tee-supplicant* process is used, e.g. to access the REE filesystem.

OP-TEE OS

The core part of OP-TEE is the OS, which provides a small-scale OS that supports the execution of signed TAs. This is also the part that gets added to the boot image, requires an additional U-Boot SPL and support in the ATF (see previous sections). The NXP-provided adaption[10] here is especially important, as it contains security-relevant modifications for the used SoC, e.g. to access the HUK. For that, it also has access to the CAAM in secure world mode, which means it can en-/decrypt blobs only meant for use in secure world.

[10] `https://source.codeaurora.org/external/imx/imx-optee-os/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To build it, set up the toolchain and execute

```
$ make CROSS_COMPILE=arm-linux-gnueabihf- CROSS_COMPILE64=aarch64-
    linux-gnu- ARCH=arm PLATFORM=imx PLATFORM_FLAVOR=mx8qxpmek
```

This will also create a folder `export-ta_arm64`, which contains a Software Development Kit (SDK) needed to build TAs and the helper process. To get verbose output about the core itself and the TAs, the debug parameters in `mk/config.mk` may be changed. There, one can set the log levels for (`CFG_TEE_CORE_LOG_LEVEL` and `CFG_TEE_TA_LOG_LEVEL`) and core debugging (`CFG_TEE_CORE_DEBUG`). Also the name of the signing key can be chosen there, using `TA_SIGN_KEY`. For productive releases, a secret private signing key in Privacy-Enhanced Mail (PEM) format has to be supplied, as for development purposes only a publicly known development key is contained in `keys/default_ta.pem`. This can be done with

```
$ ssh-keygen -t rsa -b 2048 -f key.pem -N ''
```

The private key embedded in OP-TEE OS is then derived from the public key. Currently, all TAs are signed with the same key.

One important part of the core is the secure storage implementation. Files are stored in an encrypted way on the normal REE file system under `/data/tee`, sorted by TA ID and named with a UID. They get en-/decrypted transparently with keys derived from the HUK. Two different storage implementations exist: The standard, REE file system implementation can be enabled by setting `CFG_REE_FS`, the other, using RPMB memory, by setting `CFG_RPMB_FS`. Depending on the storage ID, TAs can choose which one they want to use. A lot of additional background information on the storage implementation including RPMB can be found in Linaro Connect presentations SFO15-503[11] and LAS16-504[12]. Generally, RPMB is using a dedicated eMMC hardware partition to realize data rollback protection. This is done by sharing a symmetric key and using a counter value. Operations only succeed if they use the correct key and new counter values, which makes replaying an old message (HMAC including a lower counter value) impossible. The key is derived from the HUK, which binds the RPMB to the system. OP-TEE can be configured to program the key on first usage with config flag `CFG_RPMB_WRITE_KEY`. Also a test-key can be enabled by `CFG_RPMB_TESTKEY`. The test-key is included in `core/tee/tee_rpmb_fs.c` and as follows:

```
D3EB3EC3 6E334C9F 988CE2C0 B8595461
0D2BCF86 64844DF2 AB56E6C6 1BB701E4
```

Additionally, RPMB emulation can be enabled in the OP-TEE client for convenient testing as described in section 4.4.6. However, using the test-key or enabling automatic key writing is extremely dangerous if done in a productive environment and hence should never be done beyond the testing stage.

[11] https://s3.amazonaws.com/connect.linaro.org/sfo15/Presentations/09-25-Friday/SFO15-503-%20Secure%20storage%20in%20OP-TEE.pdf

[12] https://s3.amazonaws.com/connect.linaro.org/las16/Presentations/Friday/LAS16-504%20-%20Secure%20Storage%20updates%20in%20OP-TEE.pdf

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TA Type		Storage	Mode	Protection	U-Boot
Static	–	TOS integrated	root	●	yes
Dynamic	Early	TOS attached	user	●	yes
	REE FS	ROS file system	user	∗	no
	Sec. Storage	TOS storage	user	∗	RPMB only

● Good, ∗ Medium, ○ Bad

Table 4.3: OP-TEE TA types

Kernel Driver: `optee`

The OP-TEE kernel driver is included in the official Linux Kernel since release `4.12`. It handles the communication between the two worlds. To enable it, the flag `CONFIG_OPTEE` must be enabled when building.

Helper Process: `tee-supplicant`

The helper process is used to be able to use REE functionality in OP-TEE, e.g. ROS TA access. A corresponding NXP adaption is available[13]. When building it with

```
$ make CROSS_COMPILE=aarch64-linux-gnu-
```

two files are created: The helper application `tee-supplicant` should be copied to `/usr/bin`, whereas its corresponding library `libteec.so` should end up in `/usr/lib`. Also the created `export` directory is important, as it is needed as SDK to build TAs. For building with `gcc8`, a small change has to be made, otherwise the build will fail. The patch[14] can be found in listing 4.1. For debugging purposes, RPMB memory can be emulated by setting `RPMB_EMU`. This is initially set and located in the Makefile of `tee-supplicant`.

Trusted Application (TA)

The programs running on top of OP-TEE OS are called TAs. There exist different types, as table 4.3 shows. The two main variants are static TAs, which are basically extending the GP API implemented in OP-TEE OS. More often used are the dynamic TAs, which are more flexible and preferred due to requiring a lower level of privilege for execution. There are three different types, which differ in terms where they are stored. That also affects their protection, as TAs not stored within the OP-TEE core are prone to downgrade attacks (as an attacker can just replace a patched version with a previous, unpatched one). In terms of upgrades, they bring more flexibility, though.

[13] https://source.codeaurora.org/external/imx/imx-optee-client/
[14] https://github.com/OP-TEE/optee_client/issues/126#issuecomment-399440707

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
1  From 30dd2986fb64aba7ee78d4e231c344e2c39d7999 Mon Sep 17 00:00:00 2001
2  From: Simon Hughes <simon.hughes@arm.com>
3  Date: Thu, 21 Jun 2018 17:22:23 +0100
4  Subject: [PATCH] Fix for teec_trace.c snprintf -Werror=format-truncation=
5   error.
6
7  Signed-off-by: Simon Hughes <simon.hughes@arm.com>
8  ---
9   libteec/src/teec_trace.c | 3 ++-
10  1 file changed, 2 insertions(+), 1 deletion(-)
11
12 diff --git a/libteec/src/teec_trace.c b/libteec/src/teec_trace.c
13 index 78b79d6..c91bc43 100644
14 --- a/libteec/src/teec_trace.c
15 +++ b/libteec/src/teec_trace.c
16 @@ -106,7 +106,8 @@ int _dprintf(const char *function, int flen, int line,
       int level,
17       */
18      int thread_id = syscall(SYS_gettid);  /* perf issue ? */
19
20 -    snprintf(prefixed, MAX_PRINT_SIZE,
21 +    int len = 0;
22 +    len  = snprintf(prefixed+len, MAX_PRINT_SIZE,
23          "%s [%d] %s:%s:%d: %s",
24          trace_level_strings[level], thread_id, prefix, func,
25          line, raw);
26 --
27 2.7.4
```

Listing 4.1: OP-TEE `libteec` `gcc8` patch

Some examples are available on the official Github repository[15]. To build them, paths to OP-TEE OS and client dependencies have to be set:

```
$ make CROSS_COMPILE=aarch64-linux-gnu- TA_DEV_KIT_DIR=../imx-optee-os
    /export-ta_arm64 TEEC_EXPORT=../imx-optee-client/out/export
```

For debugging TAs, the log level can be set in the Makefile by `CFG_TEE_TA_LOG_LEVEL`. The resulting binary should be copied to `/usr/bin` and the TAs to `/lib/optee_armtz`.

Tests: `xtest`

Readily available for use, there is also a test-suite for OP-TEE, again adapted by NXP[16]. After building the test-suite with

```
$ make CROSS_COMPILE=aarch64-linux-gnu- TA_DEV_KIT_DIR=../imx-optee-os
    /export-ta_arm64 OPTEE_CLIENT_EXPORT=../imx-optee-client/out/export
```

the resulting binary `xtest` should be copied to `/usr/bin` and the TAs should end up in `/lib/optee_armtz`.

[15] https://github.com/linaro-swg/optee_examples
[16] https://source.codeaurora.org/external/imx/imx-optee-test/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Execute

```
$ sudo xtest
```

after OP-TEE is configured and started.

Checking OP-TEE

The first step after assembling the whole boot image is to check if OP-TEE is up and running at start. This can be seen by the boot output, which should look like in listing 4.2 if debugging is enabled. In Linux, the inserted device-tree node (listing 4.3) should be visible. This can be checked by examining the device-tree mapping at `/proc/device-tree`. If this node exists, it should be possible to run

```
$ sudo modprobe optee
```

to load the OP-TEE kernel module without any errors. This will now create two devices, `tee0` and `teepriv0`. If these exist, it is possible to start the helper application (in the background)

```
$ sudo tee-supplicant &
```

and afterwards run the first TA

```
$ sudo optee_example_hello_world
```

This all should look similar to listing 4.4. After that, an `xtest` run is recommended to ensure full functionality.

```
1  UD/TC:0 add_phys_mem:530 TEE_SHMEM_START type NSEC_SHM 0xffc00000 size 0
       x00400000
2  D/TC:0 add_phys_mem:530 TA_RAM_START type TA_RAM 0xfe200000 size 0x01a00000
3  D/TC:0 add_phys_mem:530 VCORE_UNPG_RW_PA type TEE_RAM_RW 0xfe04f000 size 0
       x001b1000
4  D/TC:0 add_phys_mem:530 VCORE_UNPG_RX_PA type TEE_RAM_RX 0xfe000000 size 0
       x0004f000
5  D/TC:0 add_phys_mem:530 GICR_BASE type IO_SEC 0x51a00000 size 0x00200000
6  D/TC:0 add_phys_mem:530 GICD_BASE type IO_SEC 0x51a00000 size 0x00200000
7  D/TC:0 add_phys_mem:543 Physical mem map overlaps 0x51a00000
8  D/TC:0 add_phys_mem:530 CONSOLE_UART_BASE type IO_NSEC 0x5a000000 size 0
       x00400000
9  D/TC:0 verify_special_mem_areas:468 No NSEC DDR memory area defined
10 D/TC:0 add_va_space:569 type RES_VASPACE size 0x00a00000
11 D/TC:0 add_va_space:569 type SHM_VASPACE size 0x02000000
12 D/TC:0 dump_mmap_table:702 type IO_NSEC      va 0xf9200000..0xf95fffff pa 0
       x5a000000..0x5a3fffff size 0x00400000 (pgdir)
13 D/TC:0 dump_mmap_table:702 type NSEC_SHM     va 0xf9600000..0xf99fffff pa 0
       xffc00000..0xffffffff size 0x00400000 (pgdir)
14 D/TC:0 dump_mmap_table:702 type TA_RAM       va 0xf9a00000..0xfb3fffff pa 0
       xfe200000..0xffbfffff size 0x01a00000 (pgdir)
15 D/TC:0 dump_mmap_table:702 type RES_VASPACE  va 0xfb400000..0xfbdfffff pa 0
       x00000000..0x009fffff size 0x00a00000 (pgdir)
16 D/TC:0 dump_mmap_table:702 type IO_SEC       va 0xfbe00000..0xfbffffff pa 0
       x51a00000..0x51bfffff size 0x00200000 (pgdir)
```

Continued on next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
17 D/TC:0 dump_mmap_table:702 type SHM_VASPACE  va 0xfc000000..0xfdffffff pa 0
       x00000000..0x01ffffff size 0x02000000 (pgdir)
18 D/TC:0 dump_mmap_table:702 type TEE_RAM_RX   va 0xfe000000..0xfe04efff pa 0
       xfe000000..0xfe04efff size 0x0004f000 (smallpg)
19 D/TC:0 dump_mmap_table:702 type TEE_RAM_RW   va 0xfe04f000..0xfe1fffff pa 0
       xfe04f000..0xfe1fffff size 0x001b1000 (smallpg)
20 D/TC:0 core_mmu_entry_to_finer_grained:653 xlat tables used 1 / 5
21 D/TC:0 core_mmu_entry_to_finer_grained:653 xlat tables used 2 / 5
22 I/TC:
23 D/TC:0 init_canaries:164 #Stack canaries for stack_tmp[0] with top at 0
       xfe083ab8
24 D/TC:0 init_canaries:164 watch *0xfe083abc
25 D/TC:0 init_canaries:164 #Stack canaries for stack_tmp[1] with top at 0
       xfe0842f8
26 D/TC:0 init_canaries:164 watch *0xfe0842fc
27 D/TC:0 init_canaries:164 #Stack canaries for stack_tmp[2] with top at 0
       xfe084b38
28 D/TC:0 init_canaries:164 watch *0xfe084b3c
29 D/TC:0 init_canaries:164 #Stack canaries for stack_tmp[3] with top at 0
       xfe085378
30 D/TC:0 init_canaries:164 watch *0xfe08537c
31 D/TC:0 init_canaries:165 #Stack canaries for stack_abt[0] with top at 0
       xfe080db8
32 D/TC:0 init_canaries:165 watch *0xfe080dbc
33 D/TC:0 init_canaries:165 #Stack canaries for stack_abt[1] with top at 0
       xfe0819f8
34 D/TC:0 init_canaries:165 watch *0xfe0819fc
35 D/TC:0 init_canaries:165 #Stack canaries for stack_abt[2] with top at 0
       xfe082638
36 D/TC:0 init_canaries:165 watch *0xfe08263c
37 D/TC:0 init_canaries:165 #Stack canaries for stack_abt[3] with top at 0
       xfe083278
38 D/TC:0 init_canaries:165 watch *0xfe08327c
39 D/TC:0 init_canaries:167 #Stack canaries for stack_thread[0] with top at 0
       xfe07e138
40 D/TC:0 init_canaries:167 watch *0xfe07e13c
41 D/TC:0 init_canaries:167 #Stack canaries for stack_thread[1] with top at 0
       xfe080178
42 D/TC:0 init_canaries:167 watch *0xfe08017c
43 I/TC:  OP-TEE version: 01.02.00-190118-d6-1089-90-g1f1b2777-dev #1 Mi Okt 30
        08:32:21 UTC 2019 aarch64
44 D/TC:0 tee_ta_register_ta_store:534 Registering TA store: 'REE' (priority
       10)
45 D/TC:0 mobj_mapped_shm_init:559 Shared memory address range: fc000000,
       fe000000
46 I/TC:  Initialized
47 D/TC:0 init_primary_helper:930 Primary CPU switching to normal world boot
```

Listing 4.2: OP-TEE boot debug output

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
1 firmware {
2     optee {
3         compatible = "linaro,optee-tz";
4         method = "smc";
5     };
6 };
```

Listing 4.3: OP-TEE device-tree node

```
1  $ sudo optee_example_hello_world
2  D/TC:0 tee_ta_init_pseudo_ta_session:274 Lookup pseudo TA 8aaaf200-2450-11e4
       -abe2-0002a5d5c51b
3  D/TC:0 load_elf:842 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b
       (Secure Storage TA)
4  D/TC:0 load_elf:842 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b
       (REE)
5  D/TC:0 load_elf_from_store:810 ELF load address 0x40005000
6  D/TC:0 tee_ta_init_user_ta_session:1019 Processing relocations in 8aaaf200
       -2450-11e4-abe2-0002a5d5c51b
7  D/TA:  TA_CreateEntryPoint:39 has been called
8  D/TA:  TA_OpenSessionEntryPoint:68 has been called
9  I/TA:  Hello World!
10 Invoking TA to increment 42F/TC:0 trace_syscall:128 syscall #1 (syscall_log)
11 D/TA:  inc_value:105 has been called
12 I/TA:  Got value: 42 from NW
13 I/TA:  Increase value to: 43
14
15 TA incremented value to 43
16 D/TC:0 tee_ta_close_session:380 tee_ta_close_session(0xfe0a5780)
17 D/TC:0 tee_ta_close_session:399 Destroy session
18 I/TA:  Goodbye!
19 D/TA:  TA_DestroyEntryPoint:50 has been called
20 D/TC:0 tee_ta_close_session:425 Destroy TA ctx
```

Listing 4.4: Running the `hello_world` example TA

4.4.7 BL33: U-Boot

The last stage in the boot process is the bootloader. A widely used and open-source solution for that is U-Boot[17]. A lot on that and its integration in embedded systems was written by Yaghmour et al. in [58]. Also included is the possibility to create an SPL, needed to integrate OP-TEE. The adapted release[18] provided by NXP is `2017.03`. This later turned out to be a problem, as the OP-TEE driver was not included before release `2018.07`, but more on that in section 4.6.

U-Boot is built in two stages: In the first, the configuration is created

```
$ make CROSS_COMPILE=aarch64-linux-gnu- imx8qxp_mek_spl_defconfig
```

[17] https://www.denx.de/wiki/U-Boot
[18] https://source.codeaurora.org/external/imx/uboot-imx/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

```
NXP i.MX8QXPMEK boot image stand-alone build script
TARGETS:
  help:          show this help message
  toolchain:     download and extract toolchains*
  download:      download sources*
  scfw:          provide SCFW image*
  seco:          provide SECO FW image*
  atf:           build ARM Trusted Firmware*
  optee-os:      build OP-TEE OS*
  optee-client: build OP-TEE client application
  optee-test:    build OP-TEE test application
  uboot:         build U-Boot and U-Boot SPL*
  image:         build unsigned image
  image-signed: build signed image
  all:           build dependencies (*), signed and unsigned image

You may provide additional build flags in second argument
(e.g. V=1 for verbose output)
```

Listing 4.5: Help output of the boot image build-script

in the second, the binaries are built

```
$ make CROSS_COMPILE=aarch64-linux-gnu-
```

It already contains a CAAM driver implementation to create key blobs, however, we could not make it work. Enabled with the flags `CONFIG_FSL_CAAM`, `CONFIG_CMD_BLOB` and `CONFIG_CMD_HASH`, the `blob` command using it instantly crashes the whole system.

4.4.8 Building the Boot-Image

For creating the boot image from the different parts in table 4.2, NXP provides a dedicated build utility, `mkimage`[19]. After copying all dependencies to the `iMX8QX` directory, build the final container with

```
$ make SOC=iMX8QX flash_spl_container
```

The created `flash.bin` can now be written to the system.

To make all this easier, an automatic build script was created. It takes all the steps from the last sections and creates a boot image after downloading all needed toolchains and building all needed dependencies. Additionally, it supports building the host-side programs of OP-TEE as also the developed TA. Its usage information can be seen in listing 4.5.

For running the system off an SD-card, the *Linux User's Guide* [94] shows where to write the boot image. However, also other ways exist, as using NXP's Universal Update Utility (UUU)[20] and `fastboot` in U-Boot.

[19] https://source.codeaurora.org/external/imx/imx-mkimage/
[20] https://github.com/NXPmicro/mfgtools

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Run

```
$ fastboot 0
```

to enable it and then UUU can write the bootloader to the boot partition in eMMC

```
$ sudo uuu -b emmc flash.bin
```

while the MEK is connected via USB-C.

4.5 System Authentication by NXP

To ensure system authentication, the boot image has to be signed and its signature checked. How this works is depictured in figure 4.3. The SECO ROM works a RoT, and authenticates the SECO Firmware (FW) and the SCFW. In another container, it authenticates all the other software steps up to U-Boot. The results of the AHAB implementation can be examined by including the AHAB tools, using the flag `CONFIG_AHAB_BOOT`.

The image needs to be signed and the RoT key needs to be written to the ROM. For that, NXP provides the Code Signing Tool (CST)[21] with an extra manual [95]. Contained in `releases/keys`, the interactive `ahab_pki_tree.sh` script can be found to create authenticating PKI tree. The AHAB supports four different keys, from which the first three can be revoked (marked invalid on the system). The public keys are combined to a key table, which is later included in the image. A hash of this key table is written to the fuses of the system by using the `fuse` command in U-Boot, to serve as RoT. On boot, the system loads the key table and signatures and only continues if the hash of the loaded key table matches the stored hash and the signatures are valid for one of the contained keys. An extensive step-by-step description is contained in the NXP U-Boot repository.

Creating the signed image is now a little bit more complicated as in section 4.4.8, as this needs multiple stages. First, the container including U-Boot and the ATF has to be created, again using the `mkimage` tool

```
$ make SOC=iMX8QX u-boot-atf-container.img
```

Then, this container has to be signed using the CST binary, e.g. the Linux 64-bit version contained at `release/linux64/bin`. Additionally needed is a configuration file, naming options, offsets and files which are needed in the process. Some examples of these are also contained in the U-Boot repository[22]. Execute

```
$ cst -i cst_atf_image.txt -o u-boot-atf-container-signed.img
```

to get the signed container. The second container is again created by `mkimage`

```
$ make SOC=iMX8QX flash_spl
```

and also signed by the CST using a second configuration file

```
$ cst -i cst_boot_image.txt -o flash-signed.bin
```

[21] https://www.nxp.com/webapp/Download?colCode=IMX_CST3.2.0_TOOL

[22] https://source.codeaurora.org/external/imx/uboot-imx/tree/doc/imx/ahab/csf_examples?h=imx_v2018.03_4.14.98_2.0.0_ga

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
215         echo "writing new offsets (${offset_head}, ${offset_sign}) to ${
                csf_noext}.txt"
216         sed -e "s|__OFFSETS__|${offset_head} ${offset_sign}|" ${csf_noext}.
                template > ${csf_noext}.txt
217
218         echo "writing image path to ${csf_noext}.txt"
219         sed -i -e "s|__FILE__|${image}|" ${csf_noext}.txt
220
221         echo "writing CST path to ${csf_noext}.txt"
222         sed -i -e "s|__CST_ROOT__|${CST_ROOT}|" ${csf_noext}.txt
```

Listing 4.6: Using `sed` to create configuration from templates (`build.sh`)

```
1  [Header]
2  Target = AHAB
3  Version = 1.0
4
5  [Install SRK]
6  # SRK table generated by srktool
7  File = "__CST_ROOT__/crts/SRK_1_2_3_4_table.bin"
8  # Public key certificate in PEM format
9  Source = "__CST_ROOT__/crts/SRK1_sha384_secp384r1_v3_ca_crt.pem"
10 # Index of the public key certificate within the SRK table (0 .. 3)
11 Source index = 0
12 # Type of SRK set (NXP or OEM)
13 Source set = OEM
14 # bitmask of the revoked SRKs
15 Revocations = 0x0
16
17 [Authenticate Data]
18 # Binary to be signed generated by mkimage
19 File = "__FILE__"
20 # Offsets = Container header  Signature block (printed out by mkimage)
21 Offsets = __OFFSETS__
```

Listing 4.7: Boot image configuration file template for CST

The created, signed image `flash-signed.bin` is now ready to be transferred to the system.

To use in the automatic build script, the configuration files have to be created automatically during the build. This can be done by extracting the offset values from the output of `mkimage` and then creating a configuration file based on them. How this was done can be seen in listing 4.6, using configuration templates as can be seen in listing 4.7.

After booting an image, `ahab_status` in U-Boot can be used to check the secure boot state. Examples of the output with or without a valid signature can be seen in listing 4.8 and listing 4.9. Before the device is locked, it is still able to boot, even if the check fails. However, in the end, it needs to be locked to enforce the CoT. The command

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
=> ahab_status
Lifecycle: 0x0020, NXP closed

No SECO Events Found!
```

Listing 4.8: U-Boot secure boot check passed

```
=> ahab_status
Lifecycle: 0x0020, NXP closed

SECO Event[0] = 0x0087EE00
        CMD = AHAB_AUTH_CONTAINER_REQ (0x87)
        IND = AHAB_NO_AUTHENTICATION_IND (0xEE)
```

Listing 4.9: U-Boot secure boot check failed, wrong or missing signature

`ahab_close` can be used for that. After that, it won't boot any more without an image containing a valid signature.

Now that the system is secured up to the bootloader, it has to authenticate the system image to extend the CoT further. This can easily be done by signing the image and embedding the corresponding public key in U-Boot, programming it to check the signature and enforce validity during the boot process. However, for update and file system encryption, the following section will explain our approach.

4.6 Key Management by TA

A general introduction to TAs has already been given in section 4.4.6. Here, we will now design the sHSM TA to execute the key management in the system. It will provide functions to store, load and use secret keys as also to protect arbitrary data.

4.6.1 Requirement Design Decisions

To meet our requirements, we had to make several design decisions. They are explained in the following sections.

U-Boot Usability

As the TA shall be usable already during the bootloader stage, the standard REE file system storage for TAs cannot be used, otherwise it would be ready only after booting the Linux OS and starting the OP-TEE helper application. However, as already mentioned in section 4.4.6, OP-TEE provides also other storage options. Therefore, we can make use of the (quite new) early TAs. Early TAs are embedded into the data section of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
7  #ifndef CONFIG_IMX8QX_H
8  #define CONFIG_IMX8QX_H
9
10 #define DRAM0_BASE     0x80000000
11 #define DRAM0_SIZE     CFG_DDR_SIZE
12
13 #define DRAM0_NSEC_BASE    DRAM0_BASE
14 #define DRAM0_NSEC_SIZE    (CFG_TZDRAM_START - DRAM0_NSEC_BASE)
15
16 #endif
```

Listing 4.10: Modifications to enable dynamic shared memory in OP-TEE (`imx8qx.h`)

OP-TEE OS, as described in its commit[23]. They can be included by first building the dependencies needed for TA compilation with

```
$ make [...] CFG_EARLY_TA=y ta_dev_kit
```

After that, all TAs can be built according to their Makefile and included by building the OP-TEE image with its paths as

```
$ make [...] EARLY_TA_PATHS=<paths>
```

An example for using the early TA feature is the new Android Verified Boot (AVB) TA[24].

Due to differences in the OP-TEE helper application in U-Boot, also dynamic shared memory must be supported by OP-TEE and enabled. As this is not supported in the latest official NXP release, we decided to use the latest, unofficial `imx_4.19.35_1.1.0` release. However, even if the feature is implemented in the newer release, it is not enabled. This can be done by modifying `core/arch/arm/plat-imx/config/imx8qx.h` as in listing 4.10 (adding lines 13 and 14).

In U-Boot, an implementation of the OP-TEE driver and its helper application is needed. This was added recently and featured in the YVR18-117 Linaro Connect presentation[25]. However, again the official NXP release is too old to include it, making us switch again to the newer, unofficial `imx_4.19.35_1.1.0` release based on version `2019.04` as a backport of the driver did not seem feasible. One thing we still had to add in this release was the OP-TEE node (see listing 4.3) in the Device Tree Blob (DTB) for OP-TEE to be usable from U-Boot. Also, the interface implemented by the U-Boot OP-TEE helper application for communicating with TAs turned out to be quite different to the Linux implementation. Therefore, to keep a common code base for our application used in Linux and U-Boot, we had to work around this by various defines and an extra compilation flag (more on that in the source code overview in section 4.6.2).

As the U-Boot does not have access to the REE file system, the only way to use the OP-TEE secure storage here is to go with RPMB. However, this uncovered a new problem, as

[23] `https://github.com/OP-TEE/optee_os/commit/d0c636148b3a`
[24] `https://github.com/OP-TEE/optee_os/blob/master/ta/avb`
[25] `https://s3.amazonaws.com/connect.linaro.org/yvr18/presentations/yvr18-117.pdf`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

RPMB accesses in U-Boot kept crashing the system. Using OP-TEE via Linux for doing that works flawlessly, though, making us believe there exists a problem somewhere in the U-Boot OP-TEE helper application or RPMB driver. The problem was finally located in the driver's `rpmb_route_frames` function (`drivers/mmc/rpmb.c`). Apparently DMA needs a cache aligned buffer, however, it was not aligned. A quick fix can be seen in listing 4.11, where a cache-aligned buffer is allocated (lines 409-410), data is copied (lines 427-428) before use (lines 430-459) and copied back after that (line 462). As DMA only supports 32 bytes, the size of the buffers is no real limitation here.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
404 static int rpmb_route_frames(struct mmc *mmc, struct s_rpmb *req,
405             unsigned short req_cnt, struct s_rpmb *rsp,
406             unsigned short rsp_cnt)
407 {
408   /* WORKAROUND align buffer */
409   ALLOC_CACHE_ALIGN_BUFFER(struct s_rpmb, req_frame, 32);
410   ALLOC_CACHE_ALIGN_BUFFER(struct s_rpmb, rsp_frame, 32);
411
412   unsigned short n;
413   int rc;
414
415   /*
416    * If multiple request frames are provided, make sure that all are
417    * of the same type.
418    */
419   for (n = 1; n < req_cnt; n++)
420     if (req[n].request != req->request)
421       return -EINVAL;
422
423   /* WORKAROUND copy data */
424   if (req_cnt > 32) return -EINVAL;
425   if (rsp_cnt > 32) return -EINVAL;
426
427   if (req_cnt > 0) memcpy(req_frame, req, req_cnt*sizeof(struct s_rpmb));
428   if (rsp_cnt > 0) memcpy(rsp_frame, rsp, rsp_cnt*sizeof(struct s_rpmb));
429
430   switch (be16_to_cpu(req->request)) {
431   case RPMB_REQ_KEY:
432     if (req_cnt != 1 || rsp_cnt != 1)
433       return -EINVAL;
434     rc = rpmb_route_write_req(mmc, req_frame, 1, rsp_frame, 1);
435     break;
436
437   case RPMB_REQ_WRITE_DATA:
438     if (req_cnt > 32 || rsp_cnt != 1)
439       return -EINVAL;
440     rc = rpmb_route_write_req(mmc, req_frame, req_cnt, rsp_frame, 1);
441     break;
442
443   case RPMB_REQ_WCOUNTER:
444     if (req_cnt != 1 || rsp_cnt != 1)
445       return -EINVAL;
446     rc = rpmb_route_read_req(mmc, req_frame, 1, rsp_frame, 1);
447     break;
448
449   case RPMB_REQ_READ_DATA:
450     if (req_cnt != 1 || rsp_cnt > 32)
451       return -EINVAL;
452     rc = rpmb_route_read_req(mmc, req_frame, 1, rsp_frame, rsp_cnt);
453     break;
454
455   default:
```

Continued on next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
456      debug("Unsupported message type: %d\n",
457            be16_to_cpu(req->request));
458      return -EINVAL;
459    }
460
461    /* WORKAROUND copy data back */
462    memcpy(rsp, rsp_frame, rsp_cnt*sizeof(struct s_rpmb));
463    return rc;
464  }
```

Listing 4.11: Modifications to fix the RPMB driver in U-Boot (`rpmb.c`)

Security

Not only because of U-Boot compatibility reasons, but also because of security reasons, the move for using RPMB as secure storage was made. Otherwise, if having no rollback protection in place, an attacker could possibly exchange stored keys with older ones, which he had previously copied and may got disclosed in the meantime. To use that, an eMMC supporting the special RPMB partition is required (which the MEK luckily includes). Following [48], including such a partition is mandatory for eMMC from version `4.4` on. The same is true for application storage: Due to the fact that early TAs are integrated in the boot image, rolling them back is not possible if the bootloader enforce rollback protection for updates.

Data Size

During our tests, we discovered that the maximum shared memory buffer size OP-TEE can process is set to 1MiB. Therefore, we had two choices: Configuring a bigger maximum buffer size or implementing encryption in chunks. We went with the latter approach, due to the fact that it scales to arbitrary sized data. This means, our implementation now processes all data bigger than 512KiB in chunks of 512KiB.

Development Support

As this is only a PoC implementation, it contains functions which should not be available when used in productive use, e.g. the function to read a key. They are meant for administrating while testing and debugging and shall be disabled later on.

4.6.2 Source Code Overview

The following sections will now give an overview of the different parts and functionality of our sHSM application. As mentioned before, the U-Boot OP-TEE helper application API turned out quite different to the Linux version. Therefore, we introduced the `UBOOT_TA` define for `softhsm.h` and `main.c`. If set during compilation, the application will be compiled to use the U-Boot OP-TEE interface. It can be added to the `PLATFORM_CPPFLAGS` on line 14 of the U-Boot `make` configuration `config.mk`. This

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

causes not only a different API usage, but also different include-paths, modified defines and limited functionality: Due to the fact that the file system is not yet available in U-Boot, reading and writing from and to files is not supported then. An example for the remaining differences after adding additional custom functions and macros can be seen in listing 4.12. The most noticeable difference is, that for the U-Boot implementation, the calling application has to manage the shared memory allocation on its own, whereas in the Linux implementation this is already covered by the API.

```
351 TEEC_Result shsm_data_load(shsm_ctx_t *ctx, char *id,
352         uint8_t *data, size_t *data_len)
353 {
354     TEEC_Result res;
355     uint32_t origin;
356
357     printd("loading id \"%s\"\n", id);
358
359 #ifdef UBOOT_TA
360
361     /* create shared memory buffers */
362     uint32_t shm_id_size = strlen(id);
363     TEEC_SHM *shm_id = TEEC_SHM_Create(ctx->dev, shm_id_size, id,
        shm_id_size);
364     if (!shm_id) return TEEC_ERROR_OUT_OF_MEMORY;
365
366     uint32_t shm_data_size = *data_len;
367     TEEC_SHM *shm_data = TEEC_SHM_Create(ctx->dev, shm_data_size, NULL, 0);
368     if (!shm_data) {
369         TEEC_SHM_Destroy(shm_id, NULL, 0);
370         return TEEC_ERROR_OUT_OF_MEMORY;
371     }
372
373     /* prepare parameters */
374     TEEC_Params_Create(params, TEE_PARAM_ATTR_TYPE_MEMREF_INPUT,
375                                TEE_PARAM_ATTR_TYPE_MEMREF_OUTPUT,
376                                TEE_PARAM_ATTR_TYPE_NONE,
377                                TEE_PARAM_ATTR_TYPE_NONE);
378     params[0].u.memref.shm = shm_id;
379     params[0].u.memref.size = shm_id_size;
380     params[1].u.memref.shm = shm_data;
381     params[1].u.memref.size = shm_data_size;
382
383     /* call TA */
384     res = TEEC_Invoke(ctx->dev, ctx->sess, TA_SOFTHSM_CMD_DATA_LOAD,
385             params, &origin);
386
387     /* copy data back */
388     *data_len = params[0].u.memref.size;
389     TEEC_SHM_Destroy(shm_data, data, shm_data_size);
390     TEEC_SHM_Destroy(shm_id, NULL, 0);
391
392 #else /* !UBOOT_TA */
```

Continued on next page

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
393
394     TEEC_Operation op;
395     memset(&op, 0, sizeof(op));
396     op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
397                                      TEEC_MEMREF_TEMP_OUTPUT,
398                                      TEEC_NONE,
399                                      TEEC_NONE);
400     op.params[0].tmpref.buffer = id;
401     op.params[0].tmpref.size = strlen(id);
402     op.params[1].tmpref.buffer = data;
403     op.params[1].tmpref.size = *data_len;
404
405     res = TEEC_InvokeCommand(&ctx->sess, TA_SOFTHSM_CMD_DATA_LOAD,
406                 &op, &origin);
407
408     *data_len = op.params[1].tmpref.size;
409
410 #endif /* !UBOOT_TA */
411
412     check_error(res, origin, data_len);
413     printd("%zu byte(s) loaded\n", *data_len);
414     return res;
415 }
```

Listing 4.12: sHSM API implementation: `shsm_data_load` (`softhsm.c`)

sHSM API

The sHSM API (`softhsm.h`) is wrapping the sHSM TA interface from the TA to make it readily usable for applications. To use any of the functions, a valid sHSM context has to be created. For this, the `shsm_open` and `shsm_close` functions are provided.

The first big functionality of the application is data blob handling. It provides functions to store (`shsm_data_store`), load (`shsm_data_load`), delete (`shsm_data_delete`) and list (`shsm_data_list`) arbitrary data.

The next part is secret key handling. This works slot-based, their maximum number is configurable at compile-time and separated per cryptographic algorithm (however, only AES is implemented for now). Therefore, keys can now be generated (`shsm_key_generate`), stored (`shsm_key_set`) and deleted (`shsm_key_delete`) using their corresponding slot number. Additionally, for verification purposes, also reading (`shsm_key_get`) key slots is possible. To determine which slots are used, all slots of a cryptographic algorithm can also be listed (`shsm_key_list`).

To use the stored keys, cryptographic operations are available. Giving a key-slot number to use, data can be encrypted (`shsm_encrypt_aes`) and decrypted (`shsm_decrypt_aes`) with AES. Additional functions are available to support encryption for chunks of data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
30 /**
31  * TA_SOFTHSM_CMD_DATA_LOAD - read a secure storage raw file
32  * param[0] (memref) [input]  Identifier
33  * param[1] (memref) [output] Raw data as bytes
34  * param[2] unused
35  * param[3] unused
36  */
37 #define TA_SOFTHSM_CMD_DATA_LOAD        1
```

Listing 4.13: sHSM TA interface example (softhsm_ta.h)

sHSM TA

The sHSM TA interface (softhsm_ta.h) describes the interface to the trusted world application. This interface generally consists of an integer-command-ID and up to four additional parameters, which either contain two 32-bit values or a memory reference pointer and the size of the referenced memory. As a result, their definitions are not more than a single integer with comments on the parameter conventions of the command, like can be seen in listing 4.13. The encryption was implemented using AES128 in CTR mode. Every time an encryption is started, a new Initialization Vector (IV) is randomly generated (this IV can be stored unsecured). For encryption, the function needs the encrypted data as also the IV.

sHSM Management Application

The provided management application (main.c) is using the sHSM API from above to provide all functionality via a command-line interface to the user. Its usage can be seen in listing 4.14 and is available with the help command.

In Linux, the functionality to read and write data to and from files is available. As we use AES-CTR, these files follow a custom specification to also include the randomly generated IV. For encryption, the encrypted file will be of size $blob = data + 16$ bytes, for decryption, the first 16 bytes will be interpreted as IV, leading to a size of $data = blob - 16$ bytes. The output of an example run using en- and decryption can be seen in listing 4.15 and listing 4.16, where also the IV behavior can be noted. A file plaintext.txt containing the testdata string followed by a newline byte is encrypted to a binary blob blob.bin containing the IV and the ciphertext and vice versa. This works the same in U-Boot, however, due to the not supported files, the input is read as 0x prefixed hex-encoded parameter string (instead as before as filename). The output is then dumped to stdout.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
$ sudo softhsm help
Usage:
softhsm COMMAND

available commands:
        help
        data store ID FDATA
        data load ID [FDATA]
        data delete ID
        data list
        key generate TYPE_BITS SLOT
        key set TYPE_BITS SLOT KEY
        key get TYPE SLOT
        key delete TYPE SLOT
        key list TYPE
        encrypt TYPE SLOT FDATA [FBLOB]
        decrypt TYPE SLOT FBLOB [FDATA]

supported formats:
TYPE:      aes
TYPE_BITS: aes_128, aes_192, aes_256
SLOT:      integer from 0 to maximum number configured
ID:        string identifier of max. 64 characters
FDATA:     either a file path or '0x' prefixed hex data
FBLOB:     either a file path or '0x' prefixed hex data
           first 16 bytes of content are the used IV
KEY:       hex string

Note: Files in-/output is not available in U-Boot
```

Listing 4.14: sHSM management application help message

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
$ sudo softhsm encrypt aes 1 plaintext.txt
reading from "plaintext.txt"
size (9) smaller than chunk size (524288)
D/TA:  crypt_aes:796 key_id: slot_00_01
process in one step
encrypting 9 byte(s)
D/TA:  debug_dump_key:105 dumping key info and data
D/TA:  debug_dump_key:116 objectSize: 128
D/TA:  debug_dump_key:117 maxObjectSize: 128
D/TA:  debug_dump_key:118 objectType: 0xa0000010
D/TA:  debug_dump_key:119 keySize: 128
D/TA:  debug_dump_key:120 maxKeySize: 128
D/TA:  debug_dump_key:121 objectUsage: 0xffffffff
D/TA:  debug_dump_key:122 handleFlags: 0x00030000
D/TA:  debug_dump_key:140 key: 221b368d 7f5f5978 67f52597 1f28ff75
D/TA:  crypt_aes:823 key_bits: 128
MEASUREMENT: 0.096054 seconds
00000000: 3734b69e f759cbe3 3307242b f29382ca |74...Y..3.$+....|
00000010: b5ce54e3 7512d3bf 4c                |..T.u...L|
```

Listing 4.15: sHSM encrypting data with debug output

```
$ sudo softhsm decrypt aes 1 blob.bin
reading from "blob.bin"
size (9) smaller than chunk size (524288)
D/TA:  crypt_aes:796 key_id: slot_00_01
process in one step
decrypting 9 byte(s)
D/TA:  debug_dump_key:105 dumping key info and data
D/TA:  debug_dump_key:116 objectSize: 128
D/TA:  debug_dump_key:117 maxObjectSize: 128
D/TA:  debug_dump_key:118 objectType: 0xa0000010
D/TA:  debug_dump_key:119 keySize: 128
D/TA:  debug_dump_key:120 maxKeySize: 128
D/TA:  debug_dump_key:121 objectUsage: 0xffffffff
D/TA:  debug_dump_key:122 handleFlags: 0x00030000
D/TA:  debug_dump_key:140 key: 221b368d 7f5f5978 67f52597 1f28ff75
D/TA:  crypt_aes:823 key_bits: 128
MEASUREMENT: 0.092396 seconds
00000000: 74657374 64617461 0a                |testdata.|
```

Listing 4.16: sHSM decrypting data with debug output

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 5

Evaluation

The following sections will evaluate different aspects of our implementation as also summarize findings discovered during our work.

5.1 Security

Although reaching not the same level of security as physical HSMs, our implementation using OP-TEE in the ARM TrustZone has some important benefits. Due to its interleaving of hardware and software, updates are quite easy. This helps to fulfill the requirement of crypto agility. It is easy to change to other existing algorithms or implement own ones, whereas OP-TEE also has a broad development base which is constantly implementing new features.

5.1.1 TRA

The TRA we conducted in section 3.3.3 showed different threats and risks. Based on that, in table 5.1 we evaluate which of the found risks can be mitigated by using our developed concept. The last column shows either a hyphen if our implementation is not able to provide any meaningful mitigation techniques or one of the three following mitigation strategy IDs:

1. **Mitigation against reading secret keys:** Using our PoC implementation, secret keys and certificates can now be protected inside the sHSM. This means a great protection against various online and offline threats against their disclosure. However, due to the fact that the sHSM still uses the same physical RAM (although logically protected on hardware level), it cannot help us in protecting from threat 01.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 5.1: Threat Mitigation

#	Comp.	Threat	Impact	Mit.
01	RAM	read secret keys or certificates offline	control this and other devices	-
02	RAM	destroy physically	brick system	-
03	ROM	modify FW/BL/OS offline	inject malware	2
04	ROM	modify FW/BL/OS offline	reuse hardware	2
05	ROM	modify logs	hide attack	-
06	ROM	modify FW/BL/OS offline	extract system information	2
07	ROM	read secret keys or certificates offline	control this and other devices	1
08	ROM	read FW/BL/OS offline	explore potential vulnerabilities	-
09	ROM	modify FW/BL/OS offline	brick system	2
10	ROM	destroy physically	brick system	-
11	ROM	modify FW/BL/OS offline	control this device	2
12	OS	load own BL/OS	control this device	2
13	OS	modify FW/BL/OS	inject malware	2
14	OS	modify FW/BL/OS	reuse hardware	2
15	OS	modify logs	hide attack	-
16	OS	read secret keys or certificates from ROM	control this and other devices	1
17	OS	read secret keys or certificates from RAM	control this and other devices	1
18	OS	read FW/BL/OS	explore potential vulnerabilities	-
19	OS	modify FW/BL/OS	extract system information	2
20	OS	modify FW/BL/OS	brick system	2
21	User	modify RAM via SC	control this device	-
22	User	modify BL/OS update	control this and other devices	3
23	User	use old BL/OS update	restore vulnerable software	3
24	User	read secret keys or certificates from RAM via SC	control this and other devices	-
25	User	read BL/OS update	explore potential vulnerabilities	3

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. **Mitigation against software modifications:** Due to the CoT enforcement, no external or modified software can be run on the system, as for that they need to be signed by the secret development key only owned by the manufacturer of the system. Therefore, attackers get no chance in completely overtaking the hardware or hiding malicious code in the system software. This also prevents bricking attempts of the device via software.

3. **Mitigation against system update modifications:** Update authentication can be implemented independent from our sHSM, due to the fact that this only requires to check a signature with a public key. However, to ensure authenticity, a working CoT is needed, as otherwise the attacker could simply swap the public key for update checking with its own. Additionally, with the help of encrypted updates possible by storing a symmetric key in the sHSM, also attempts of information gathering using software updates can be countered.

Especially noticeable here is also the defense-in-depth strengthening throughout nearly all threats from 12 to 20 against privileged attackers. However, not all determined risks can be mitigated by our PoC. Due to its limitations, it does not help us with

- **Threat 01: Offline extraction of secret keys or certificates from RAM**
  Due to the TEE limitations mentioned in mitigation ID 1, such attacks (e.g. a Cold-Boot attack) cannot be countered.
- **Threat 21, 24: Online side-channel attacks against RAM**
  These threats are quite similar to threat 01 and cannot be countered by any features of our concept due to the limitations of TEEs. Although logically protected on hardware level, TEEs cannot counter attacks exploiting physical weaknesses.
- **Threats 08, 18: Reconnaissance via reading software from RAM/ROM**
  As neither RAM nor ROM are encrypted, they can still be read by an offline or online attacker. This means it does not hinder attackers in disassembling the system and learning about its used software.
- **Threats 02, 10: Prevent physical destruction**
  In general, there is little one can do against intentional destruction of devices with physical access. Also our concept provides no measures against that.
- **Threats 05, 15: Protect logs from modifications**
  Although logs may be stored in an encrypted file system supported by secure boot and the sHSM, a defense-in-depth does not exist here and attackers with high enough privileges can modify log files. However, it prevents offline attacks targeting log file modifications.

Some of the mentioned threats may be impossible to address in general, for others there may exist different solutions. If the additional effort is worth the improvement must be decided on a per-threat basis. Otherwise, the risk of the remaining threats have to be accepted.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

5.1.2 IEC 62443

For the IEC 62443 evaluation, there is to say, that the outcome assumes that things outside the scope of this work are done right. There still exist millions of possibilities to wreck your system's security which would also influence the outcome here (e.g. implementation errors). However, if done right, figure 5.1 shows, that with the solution presented in this work the targeted SL 3 can be reached and even excelled. Table 5.2 shows the improved evaluation values based on the following re-evaluation of the relevant CRs and EDRs from the requirements in section 3.2:

- CR 1.5 Authenticator Management
  RE (1) Hardware security for authenticators
  As our sHSM is able to store arbitrary data, it can support protection of authenticators of any kind with hardware measures.

- CR 1.9 Strength of public key-based authentication
  RE (1) Hardware security for public key-based authentication
  As our sHSM can be used to support public key cryptography with hardware measures, the RE is fulfilled here.

- CR 1.14 Strength of symmetric key-based authentication
  RE (1) Hardware security for symmetric key-based authentication
  As our sHSM can be used to support symmetric key cryptography with hardware measures, the RE is fulfilled here.

- EDR 3.2 Protection from malicious code
  As the concept implements a CoT which protects the system software, it is impossible for attackers to insert or run their own code. This fulfills the first EDR here.

- EDR 3.10 Support for updates
  RE (1) Update authenticity and integrity
  Using the implemented CoT, the authenticity and integrity of updates can be easily ensured by signing them. Without a CoT, the attacker may be simply able to exchange the checking key.

- EDR 3.11 Physical tamper resistance and detection
  RE (1) Notification of a tampering attempt
  As the SoC includes dedicated tampering detection pins, anti-tampering measures can be integrated easily. The notification about tampering attempts is out of the scope of this work and therefore seen as "done right".

- EDR 3.12 Provisioning product supplier roots of trust
  The product supplier is responsible for provisioning the initial RoT. This is done by writing the system key hash to the ROM and closing the bootloader afterwards.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- EDR 3.13 Provisioning asset owner roots of trust
  The asset owner is able to extend the existing RoT by leveraging the sHSM functionality and adding an additional secret key, which can be used to check further software parts.

- EDR 3.14 Integrity of the boot process
  RE (1) Authenticity of the boot process
  As a full CoT is established, the integrity and authenticity of the boot process is always ensured, as no other software except signed ones can be started. Note that confidentiality is not ensured (for that, encrypted flash would be needed).

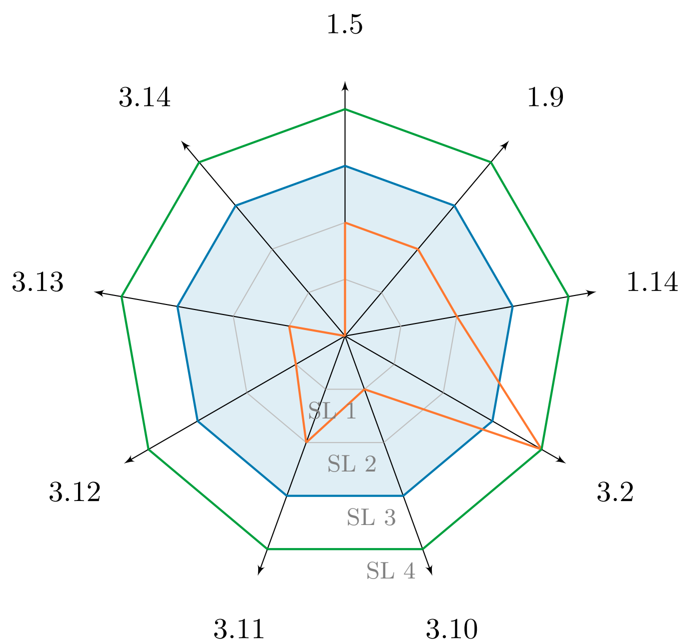

Figure 5.1: IEC 62443 potential requirement improvements

	CR			EDR					
	1.5	**1.9**	**1.14**	**3.2**	**3.10**	**3.11**	**3.12**	**3.13**	**3.14**
SL Before	2	2	2	4	1	2	1	1	0
SL After	4	4	4	4	4	4	4	4	4

Table 5.2: IEC 62443 potential requirement improvements

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```c
12 #ifndef __MEASURE_H__
13 #define __MEASURE_H__
14
15 #include <stdlib.h>
16 #include <stdio.h>
17 #include <time.h>
18
19 clock_t start;
20
21 /**
22  * @brief Start the measurement.
23  */
24 void measure_start()
25 {
26 #ifdef MEASURE
27     start = clock();
28 #endif
29 }
30
31 /**
32  * @brief End the measurement.
33  * @return Duration in seconds.
34  */
35 double measure_end()
36 {
37     double duration = 0;
38 #ifdef MEASURE
39     clock_t end = clock();
40     duration = ((double)(end - start)) / CLOCKS_PER_SEC;
41 #endif
42     return duration;
43 }
44
45 /**
46  * @brief End and print the measurement.
47  */
48 void measure_print()
49 {
50 #ifdef MEASURE
51     double duration = measure_end();
52     printf("MEASUREMENT: %f seconds\n", duration);
53 #endif
54 }
55
56 #endif /* __MEASURE_H__ */
```

Listing 5.1: Timing measurement header (`measure.h`)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5.2 Performance

A common saying goes, "there is no such thing as a free lunch". Also security always comes at a cost, most often, this is a decrease in performance. We expect a performance drawback of our implementation due to the fact, that changing the processor context to the secure TrustZone with data copying and validation on its boundaries will likely take the processor some additional time. To check our expectation and get an impression of expected performance loss, we checked the AES encryption performance of our sHSM implementation against a pure-software OpenSSL implementation with timing measurements.

This was done using AES128-CTR with different data sizes, starting from an AES block size of 16 bytes up to a (for embedded systems quite big) data size of 100MiB, and also includes the generation of a random, 16 byte initialization vector. As described in section 4.6.1, the chunk size of our TA was set to 512KiB, whereas OpenSSL does (at least seem to) process the whole buffer at once. For the measurement itself, we used simple timing functions (listing 5.1), which we inserted after preparing input and output buffers, but before creating the sHSM or OpenSSL context and after destroying the context. We simply used dd to create random test-data of different sizes, e.g.

```
$ dd if=/dev/random of=data1KiB bs=1K count=1
```

and a script for semi-automated measurement with multiple runs. The results can be seen in figure 5.2 and figure 5.3, where the first figure shows the absolute amount of time needed for encryption and the second figure the time per 16 byte-sized block. Please note that the plots have logarithmic y-axis.

The absolute values for a single encryption in figure 5.2 show a significant difference with small amounts of data. OpenSSL takes about 350µs to finish, whereas our implementation needs about 10,700µs on average, this is a factor of over 30 times slower. Another thing to note is, that it barely makes a difference if the processed data is 16 bytes or 100KiB big. Only if processing data bigger than 100KiB, we slowly start to get an increase in processing time. This is about the same for both implementations and hints about the general setup and processing time, which is independent of the amount of data. The difference in performance gets lower the bigger the processed data gets, with 100MiB our implementation taking about 686ms is only about a factor of 2.4 times slower than the OpenSSL reference with about 290ms.

```
1 #!/bin/bash
2
3 for i in {1..10}; do
4     softhsm encrypt aes 0 ${1} out.bin | grep 'MEASUREMENT:' >> ${1}.log
5 done
```

Listing 5.2: Script for conducting multiple timing measurements (test.sh)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

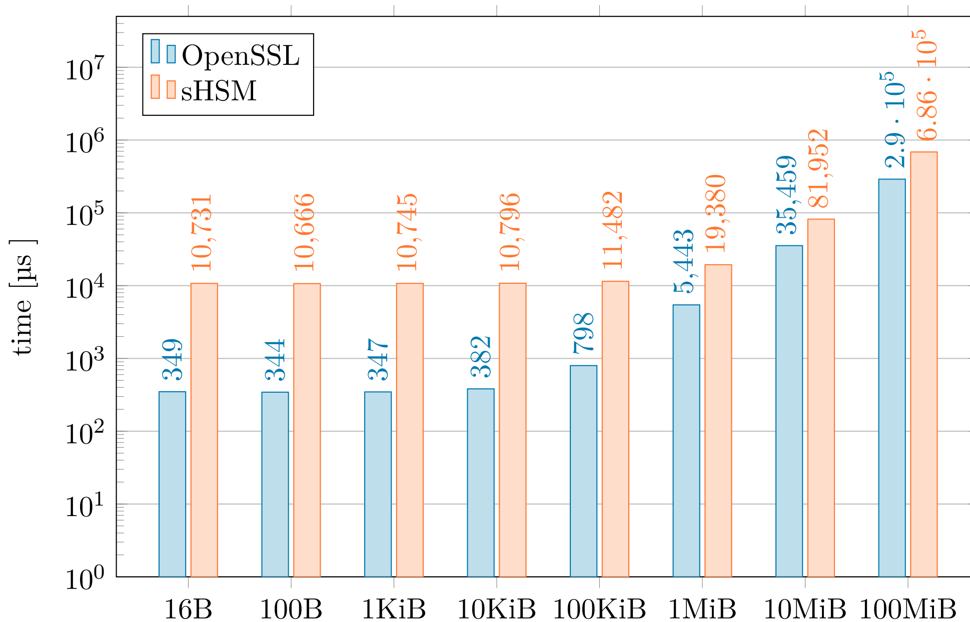

Figure 5.2: AES128-CTR encryption performance depending on data size (absolute)

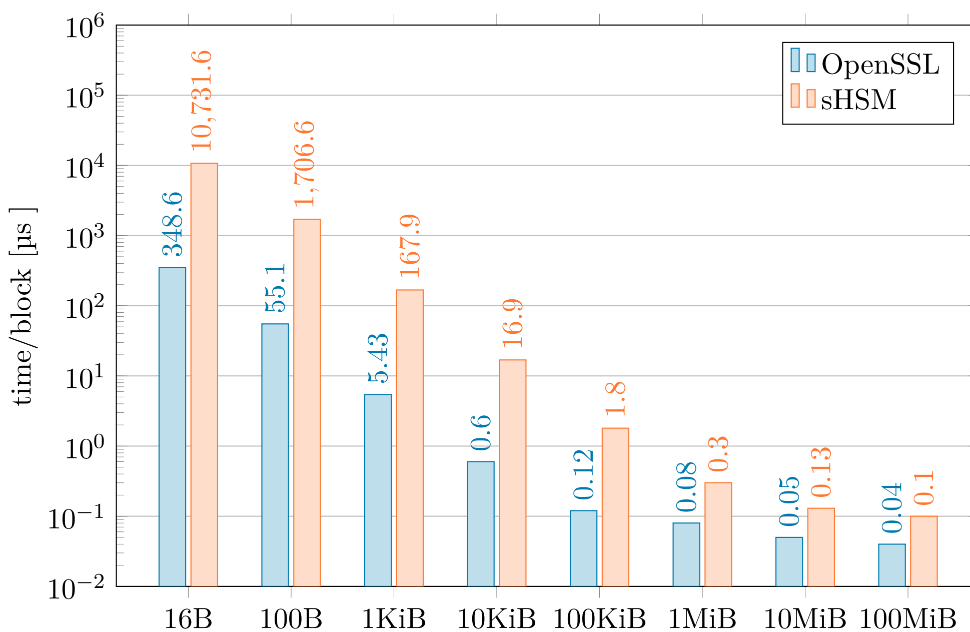

Figure 5.3: AES128-CTR encryption performance depending on data size (per block)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

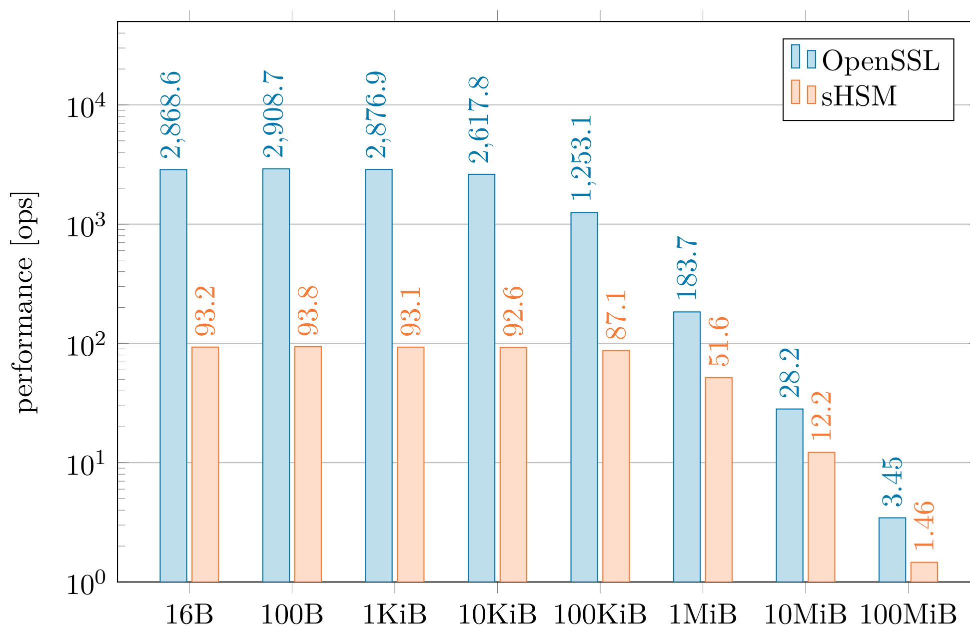

Figure 5.4: AES128-CTR encryption operations per second depending on data size

This trend can also be seen in the encryption duration per block in figure 5.3. The block time rapidly improves up to a data size of 100KiB, then slows down and will eventually become constant with bigger data sizes than we tested with. Also the narrowing performance gap in between the two implementation can be seen.

Overall, this means that the performance drawback is quite high when processing a lot of small data and reasonable when processing few small data or a lot of big data. This can also be seen in figure 5.4, which shows the achievable encryption operations per second for different sizes of data. An additional thing to add here is, that the baseline processing time of about 10.7ms we can see in figure 5.2 is likely the overhead for any TA invocation, quite independent of its function. This may be useful to keep in mind for further TA projects.

5.3 Services

The following sections will give an overview, how our service requirements defined in section 3.2.1 are met.

5.3.1 Administration & Sensor Interfaces

As for the front-end and back-end administration and sensor interfaces, these all build upon OpenSSL. OpenSSL provides an extension API which can be used for routing its cryptographic functions through a hardware layer. This can also be used here, either

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

by adding the native extension interface to the sHSM or by leveraging the existing PKCS#11[1] extension of OpenSSL.. PKCS#11 is also explained in [30] and denotes a basic and widely used API for cryptographic operations. However, due to the limited scope of this work, none of those were actually implemented in our PoC.

5.3.2 V2X Communication

V2X communication is secured by a PKI, using certificates on RSA and ECC basis. The management and operations of these certificates can be handled by our concept easily, however, again due to the limited scope, this is also still up for future implementation. For now, the sHSM only supports symmetric crypto-operations.

5.3.3 Secure Storage & Key Management

As the sHSM was designed for key management, this task can easily be handled. Using this, also encrypted and therefore confidential updates and encrypted file systems are possible now, all of them protected by the CoT implementation of secure boot and their keys protected by the sHSM.

5.4 Moving to Production

After finalizing the design and implementation, in future one typically wants to bring up a usable and secure solution into production and on custom hardware. However, due to the high interconnectedness in between hardware and the low-level software, there are some things to keep in mind when doing the transition. To use this software stack on custom hardware, the SCFW may need some modifications. For that, a porting kit[2] is provided by NXP, which contains the source code of the SCFW. Build it using

```
$ make CROSS_COMPILE=arm-none-eabi- qx R=b0
```

after making your changes. For OP-TEE, also some porting guidelines exist, the most important part covering the HUK. The OP-TEE secure storage functionality builds upon the usage of the HUK, however, due to sparse documentation and numerous existing vendors, this key is only stubbed in the original OP-TEE implementation and therefore not secure. This must be changed when planning to use it in production. The NXP fork of OP-TEE already contains the needed changes to read the SoC ID and HUK, which can be found in `core/arch/arm/plat-imx/imx-ocotp.c` and `core/arch/arm/plat-imx/imx-huk.c`. At least that is what we thought. However, after thoroughly checking, we determined that the read HUK is always 0, which has, nonetheless to says, big security implications. As the HUK is the anchor of all software

[1] https://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html

[2] https://www.nxp.com/webapp/Download?colCode=L4.14.98_2.0.0_MX8QXP&appType=license

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

trust here, this nullifies the security of all work done if not fixed before moving to production.

As a lot of key material is involved, the steps here summarize what must be done during development and manufacturing process. This starts with the steps which only have to be done once:

- Create bootloader signing keys
- Create OP-TEE signing key

Initially, a set of four system keys must be created, which are the ones used as RoT by the system. They have to be stored securely, as also has to be the OP-TEE key. This is even more important here as the keys are shared over all devices and potential loss of these keys may have catastrophic consequences. The following steps need to be done for every device during the manufacturing process:

1. Flash manufacturing boot image
2. Write bootloader signing key table hash to fuses
3. Write RPMB key to eMMC
4. Store initial keys in sHSM
5. Close bootloader
6. Flash final boot image

After writing the initial boot image in the first step, the system key hash has to be written into the fuses and the bootloader has to be closed (by burning a fuse with the AHAB commands). Also the RPMB key has to be set, this can be done by an extra OP-TEE development image contained in the initial boot image, where automatic key programming is enabled. It will derive the RPMB key from the HUK and set it. Now is also the time for storing an update-key into the sHSM. In the end, the final bootloader, stripped from development and manufacturing functionality, can be flashed.

5.5 Development Takeaways

To summarize the knowledge gained during our analysis and implementation, this section will offer some takeaways. This will also include some observations of potential problems and pitfalls we encountered during this work.

5.5.1 Development Process

During setup and the development of our PoC on the NXP evaluation kit, we stumbled upon numerous quirks or missing information pieces. Some of them are worth mentioning here.

System Complexity

Although a lot of documentation exists, the topic is just so big and the i.MX8QXP SoC features so numerous, that it takes a lot of time to get an idea of all topics. Some features,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

as the CAAM, are described in several hundred pages, and OP-TEE itself is also a huge topic on its own. As documentation tends to be scattered all over different sources, that doesn't make it easier.

Flashing the Bootloader

Flashing the boot image sounds easy, right? Well, there are some things to know or it will cause some serious headache for developers. One of these things is, that after flashing a new boot image, the system needs a hard reset (cutting the power). A soft reset via push-button is not enough and will result in loading the old boot image again. Another one is to beware about the dedicated boot partitions on the integrated eMMC. When using the `uuu` tool with destination eMMC for flashing the boot image, it will write the image to one of the hardware boot partitions. These get searched for a boot image first when the device is started, and therefore later writing a boot image to the user partition (e.g. via TFTP) seems to have no effect, as still the old bootloader from the hardware boot partition is started.

The Role of ATF for OP-TEE

When first integrating OP-TEE into our boot image, it did not work. OP-TEE was simply not started at all, however, the system booted as usual. By a hint of NXP, we figured out that for integrating OP-TEE we also need to rebuild the ATF with an additional parameter `SPD=opteed`, also described in section 4.4.5. Later we learned, that every secure payload that is meant to be started during the boot process by ATF needs some kind of special support module in the ATF, a secure payload dispatcher, here named `opteed`.

TAs in Linux and U-Boot

The development of TAs turned out not to be complicated at all, mainly due to the well documented and standardized internal GlobalPlatform interface (GPD-SPE-010 [41]) implemented by OP-TEE. To communicate with the TA from Linux and U-Boot with the same code turned out to be more tricky, as the implemented external TA interface by the OP-TEE Linux helper application and the OP-TEE U-Boot driver differs a lot. This starts with different functions and ends with missing definitions. Additionally, we found no documentation for the (quite new) U-Boot OP-TEE driver interface implemented in the U-Boot repository in `include/tee.h`. Therefore, it takes quite some work to use the same application in U-Boot and Linux to communicate with the TA. Our approach on that is explained in section 4.6.2.

5.5.2 State of Provided Software

The software stack described in section 4.4 is mainly open-source and forked by NXP to use the features of their SoCs. While working with this software, we found numerous things worth mentioning or important to note for the state of security.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CAAM Driver

Despite numerous tries, we were not able to get the CAAM module working in U-Boot. It simply kept crashing the whole system whenever we tried to access it, even when just attempting to run the integrated self-test. This led us to the conclusion, that the U-Boot CAAM driver may still need some additional work by NXP to be compatible to the i.MX8QXP SoC (however, there cannot be found any statements from NXP yet).

U-Boot Support for OP-TEE

The latest, officially supported U-Boot version by NXP turned out to be too old to contain the needed and recently added OP-TEE driver. As backporting seems too much of an effort due to the deep integration, we decided to change to the latest unofficial version by NXP, which includes the driver. However, it still did not work out of the box. Only after we modified its device-tree file to include the OP-TEE node, OP-TEE was able to determine that we use compatible hardware and could be used. Our fix is covered in section 4.6.1.

OP-TEE Support for U-Boot

For OP-TEE, it was kind of the same. The OP-TEE driver in U-Boot builds on a feature called dynamic shared memory, and the latest officially supported version by NXP did not yet include that. However, switching to the latest unofficial version of NXP did the trick and enabled us to use OP-TEE from U-Boot. But here again, the transition to the latest version was not enough. Despite supporting dynamic shared memory, it was not enabled due to an undefined secure DRAM base. We fixed that as can be seen in section 4.6.1.

OP-TEE Security

As already mentioned in section 5.4, OP-TEE has some interfaces to the hardware which are not implemented by the open-source project. These include secure hardware specific functions, which the integrator or SoC vendor has to tailor to their needs to make the system secure. One of these interfaces is reading the HUK, which is the base for all device-specific cryptographic functions. NXP did the work and implemented the needed functions. However, to our big surprise, the HUK turned out to be read as 0 all the time! As this undermines the whole security concept of OP-TEE by its core, this is important to fix. We figured out that the HUK is protected by the CAAM, so it cannot easily be read by normal-world software. The problem here tracks back to a not enabled CAAM implementation in OP-TEE (which we also weren't able to activate on our own, like in U-Boot).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

U-Boot RPMB Driver

The U-Boot RPMB driver implementation turned out to be another part that needed some work. Every time we tried to access the RPMB, it crashed the system. It turned out to be a problem in the U-Boot RPMB driver implementation which causes the issue. The DMA function of the SoC apparently wants cache-aligned data, however, this was not honored by the driver implementation. A fix for that is mentioned in section 4.6.1.

Writing the RPMB Key by OP-TEE Client

If using a properly configured manufacturing version of OP-TEE, it will write the RPMB key to the RPMB partition in eMMC on first use (never enable this in production, as it is inherently insecure). Therefore, it takes the eMMC ID and the HUK and combines them to the RPMB key. Surprisingly again, we discovered that the key written from Linux and U-Boot differ (which makes it unable to share the RPMB partition by both). For development, this can be circumvented by using the hard-coded test key, which however is obviously no solution for productive use. Tracking down the cause led to the OP-TEE helper application and the driver implementation in OP-TEE. OP-TEE, depending if running beside Linux or U-Boot, uses their help to communicate with the eMMC to spare the footprint and hassle of their own implementation. However, the eMMC ID read differed due to the fact that no endian translation was done in the U-Boot OP-TEE driver. This needs to be at least kept in mind for productive use (even better, fixed), however, for development purposes it is possible to use the provided test-key.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6

Summary

In this work, we aimed at improving the security of next-generation Collaborative Intelligent Transport Systems (C-ITS) stations by implementing a key management solution with hardware-security support. For that, we went through several stages.

6.1 Requirements and Analysis

We first started with analyzing the system and services to determine our requirements, using the IEC 62443 as a basis. Additionally, a conducted Threat and Risk Analysis (TRA) helped us to rate later security improvements. To get an idea what secure hardware module options are available, we conducted a market survey. Even though the number of available hardware is limited, we were able to examine two dedicated secure hardware modules, an NXP Hardware Security Module (HSM) and an Infineon Trusted Platform Module (TPM), as also four System on Chips (SoCs) from Autotalks, Infineon and NXP with integrated HSMs or hybrid security modules. The information policy of some companies turned out to be a problem here, as often there could not be found much information. However, NXP sticks out with much available information and an extensive and open-source software stack. It turned out that all secure hardware modules at least support the cryptographic algorithms needed and most claim crypto agility. Additionally, all SoCs provide some kind of Trusted Execution Environment (TEE) for flexible and secure Trusted Application (TA) implementations.

6.2 Concept and Implementation

The concept was developed and implemented on an NXP i.MX8QXP Multisensory Enablement Kit (MEK) and consists of a secure boot setup using U-Boot and one of five examined key management approaches. Due to its advantages in terms of flexibility, openness and costs at only a minor sacrifice in security, we chose a TEE based approach

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and implemented a Soft-Hardware Security Module (sHSM) as TA in Open Platform Trusted Execution Environment (OP-TEE). This turned out to work quite well, as OP-TEE implements a standardized GlobalPlatform (GP) interface and due to the standardized Advanced RISC Machine (ARM) TrustZone, it works on every newer ARM processor. The more complicated thing was to make the sHSM usable from U-Boot and Linux, as the APIs differ and the support for OP-TEE is quite new in U-Boot.

6.3 Findings

Using our concept, we were able to meet most of the requirements and gain significant TRA improvements. Also the applicable Component Requirements (CRs) and Embedded Device Requirements (EDRs) of IEC 62443 turned out to be fulfilled up to the highest Security Level (SL) of 4. And due to the fact that we built our key management upon a TA, flexibility and crypto agility are excellent. To make the transition to productive use easier, we summarized all tasks which need to be completed before that and all steps concerning cryptographic material which are needed for manufacturing setup and on every device during the process.

As security is not free in terms of resources, we saw an expected performance drop, although the varying impact was surprising. We conducted timing measurements of encrypting different sized data amounts with AES128-CTR against OpenSSL. It turned out, that up to about 100KiB, the static overhead dominates the dynamic one, leading to nearly constant measurements and a constant performance drop of factor 30. With increasing amounts of data, the dynamic overhead begins to dominate, which leads to a narrowing gap an a performance drop of about factor 2.4 for our biggest measured data size of 100MiB. This means, our solution works quite well for big data sizes, however, processing many small messages turned out to be challenging. As the overhead is tied to the underlying OP-TEE, there likely does not exist much room for improvements.

During our development, we discovered several things worth noting, as also some implementation shortcomings. The NXP i.MX8QXP is a quite powerful and feature-rich, but also complex system. However, the provided documentation is freely available and covers most of the important parts with reasonable depth. Also OP-TEE with its implemented GP standard provides good documentation. Nevertheless, the feature of using TAs in U-Boot is quite new, little documented and still needs some work. To be able to use it, we had to move from the latest officially supported release of NXP provided software (forks of established open-source projects) to the latest unofficial one, as the other was too old to include our needed features. In general, the state of provided software turned out well, but including some flaws. The Cryptographic Assertion and Assurance Module (CAAM) is not working in U-Boot and we had to fix some minor things to get U-Boot and OP-TEE working together as expected. However, one thing stands out as it undermines the whole OP-TEE security: Reading the Hardware Unique Key (HUK) returns 0, as there exist only a non-working implementation by NXP. This is important, as the HUK

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

functions as basis for nearly all cryptographic features of OP-TEE and should be, as the name hints, unique per device.

In the end, we managed to gather a lot of information and extended our knowledge to build upon in future. Although we did not get as far as expected, we hope this work can serve as an introduction and example for securing embedded systems and especially using the upcoming i.MX8 SoC family.

6.4 Future Work

In general there is to say, that we will likely see more work on C-ITS in future. Vehicle-to-everything (V2X) communication is just about to transition into the daily life of millions of people and more and more devices supporting it will be released. Security of these devices will – also because of the safety aspect – therefore have a huge impact on people's life. As a solid basis, existing research for embedded systems and Internet of Things (IoT) security can be used, however, some topics need specific tailoring.

In the scope of our work, there are still some topics which have work to be done. Our sHSM implementation was only intended as Proof of Concept (PoC) and is nowhere near finished, as there are numerous additional features which would be useful and some that are needed for our requirement fulfillment. The first would be the extension to ECC and RSA algorithms, which are needed for message signing functionality. This was not implemented yet due to time constraints, but should be easy to achieve because of the flexible TA design. Another thing still needed to make the sHSM usable by many established services is an interface to OpenSSL, either implemented natively or as Public Key Cryptography Standard #11 (PKCS#11) interface. An interesting proposition for that was made in the Linaro Connect presentation HKG18-402[1], where a mapping of PKCS#11 to OP-TEE functions is introduced. The final thing we consider worth implementing is access control for normal world binaries. Currently, every program with `root` permissions is allowed to call and use arbitrary TAs. The idea here is now to implement an identity check enforced by the OP-TEE driver, to allow using a TA without `root` permissions and only for explicitly specified binaries. This may not help us against privileged attackers, however, it would prevent common programs to access TAs not intended for them.

As we were not able to provide fixes for all shortcomings we discovered, this is some more work which needs to be done before using the affected features. The most important part here is to enable the CAAM implementation in OP-TEE, so that the HUK can be read correctly. Currently, even if it seems that the feature is implemented, the HUK is read as 0. Therefore, everything OP-TEE related is inherently insecure in this state. The other part, also about the CAAM, is fixing the CAAM driver for U-Boot, so the key management approach using CAAM can be examined in depth. This may be interesting because of the even higher security level it can provide.

[1] `http://connect.linaro.org.s3.amazonaws.com/hkg18/presentations/hkg18-402.pdf`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

The last thing on our list is leveraging OP-TEE for ITS-G5 messages. Current regulation approaches often demand a secure hardware module for signing and checking the signatures of the messages, however, they have not settled yet if they will consider a software module as sufficient. However, anyhow, there exists little knowledge in this direction and the topic is for sure worth further examination.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Figures

2.1 Standard boot flow 12
2.2 ARMv8 AArch64 boot flow 13
2.3 Chain of Trust example on generic secure boot flow 15

3.1 System model for TRA 25
3.2 NXP SXF1800 HSM block diagram (`www.nxp.com`) 29
3.3 Infineon OPTIGA TPM software and features [80] 30
3.4 Autotalks Craton 2 block diagram (`www.elektronikpraxis.vogel.de`) 31
3.5 Infineon Auric TriCore security diagram (`www.infineon.com`) 32
3.6 NXP i.MX8 SoC features (`www.nxp.com`) 33
3.7 NXP i.MX8 SoC features (`www.nxp.com`) 34

4.1 NXP i.MX8QuadXPlus MEK 40
4.2 System authentication and secure services concept 45
4.3 NXP i.MX8QXP boot flow and authentication 47
4.4 OP-TEE architecture overview (`www.linaro.org`) 49

5.1 IEC 62443 potential requirement improvements 73
5.2 AES128-CTR encryption performance depending on data size (absolute) . 76
5.3 AES128-CTR encryption performance depending on data size (per block) 76
5.4 AES128-CTR encryption operations per second depending on data size . 77

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

2.1 ARMv8 privilege execution levels (adapted from [56]) 13

3.1 IEC 62443 security levels 20
3.2 Threat Analysis 27
3.3 Risk Analysis 28

4.1 Comparison of key-handling approaches 42
4.2 Boot-image dependencies 46
4.3 OP-TEE TA types 51

5.1 Threat Mitigation 70
5.2 IEC 62443 potential requirement improvements 73

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Listings

4.1 OP-TEE libteec gcc8 patch 52
4.2 OP-TEE boot debug output 53
4.3 OP-TEE device-tree node 55
4.4 Running the hello_world example TA 55
4.5 Help output of the boot image build-script 56
4.6 Using sed to create configuration from templates (build.sh) 58
4.7 Boot image configuration file template for CST 58
4.8 U-Boot secure boot check passed 59
4.9 U-Boot secure boot check failed, wrong or missing signature 59
4.10 Modifications to enable dynamic shared memory in OP-TEE (imx8qx.h) 60
4.11 Modifications to fix the RPMB driver in U-Boot (rpmb.c) 62
4.12 sHSM API implementation: shsm_data_load (softhsm.c) 64
4.13 sHSM TA interface example (softhsm_ta.h) 66
4.14 sHSM management application help message 67
4.15 sHSM encrypting data with debug output 68
4.16 sHSM decrypting data with debug output 68
5.1 Timing measurement header (measure.h) 74
5.2 Script for conducting multiple timing measurements (test.sh) . . . 75

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

AHAB Advanced High Assurance Boot. 34, 46, 57, 79

ARM Advanced RISC Machine. 10–14, 31, 33–35, 37, 43, 44, 48, 69, 84, 93

ATF ARM Trusted Firmware. 13, 48, 49, 57, 80

AVB Android Verified Boot. 60

C-ITS Collaborative Intelligent Transport Systems. xi, xiii, 1–4, 7–9, 11, 19, 23, 83, 85

C-V2X Cellular V2X. 31

C2C Car-to-Car. 29

CAAM Cryptographic Assertion and Assurance Module. 33–35, 40, 42–45, 49, 56, 80, 81, 84, 85

CC Common Criteria. 9, 29, 30, 36

CERT Computer Emergency Response Team. 16

CoT Chain of Trust. 9, 14, 39, 45, 58, 59, 71–73, 78

CPS Cyber-Physical Systems. 4, 9, 16, 19

CR Component Requirement. 20, 22, 72, 84

CSRA Cyber Security Research Alliance. 9

CST Code Signing Tool. 57

DFD Data-Flow Diagram. 16

DH Diffie-Hellman. 23, 24

DMA Direct Memory Access. 41–43, 61, 82

DREAD Damage, Reproducibility, Exploitability, Affected Users, Discoverability. 16, 19, 21, 25

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

DTB Device Tree Blob. 60

ECC Elliptic Curve Cryptography. 29, 36

EDR Embedded Device Requirement. 20, 22, 72, 84

eMMC embedded Multi-Media Card. 12, 45, 50, 57, 63, 79, 80, 82

ETSI European Telecommunications Standards Institute. 1

EU European Union. 8

EVITA E-safety Vehicle Intrusion Protected Applications. 9, 11, 32, 36

FIPS Federal Institute Processing Standards. 29, 36

FPGA Field-Programmable Gate Array. 14

FR Foundational Requirement. 20

fTPM firmware Trusted Platform Module. 44

FW Firmware. 57

GP GlobalPlatform. 11, 48, 51, 84

HAB High Assurance Boot. 33, 36

HSM Hardware Security Module. 10, 15, 29, 31, 32, 35–37, 40, 43, 69, 83

HUK Hardware Unique Key. 45, 49, 50, 78, 79, 81, 82, 84, 85

IACS Industrial Automation Control Systems. 4, 20

IEEE Institute of Electrical and Electronics Engineers. 1

IoT Internet of Things. xi, xiii, 1, 2, 4, 8, 9, 11, 19, 85

ITS Intelligent Transport Systems. 7, 8

IV Initialization Vector. 66

MEK Multisensory Enablement Kit. 39, 47, 57, 63, 83

MMU Memory Management Unit. 13

MTM Mobile Trusted Module. 10, 14

NDA Non-Disclosure Agreement. 35

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

NIST National Institute of Standards and Technology. 29

OP-TEE Open Platform Trusted Execution Environment. 11, 14, 33, 34, 37, 44–46, 48–53, 55, 56, 59–61, 63, 69, 78–82, 84–86

OS Operating System. 8–10, 14, 15, 25, 26, 39, 44, 45, 48–52, 59, 60

OTP One-Time Programmable. 42

PEM Privacy-Enhanced Mail. 50

PKCS#11 Public Key Cryptography Standard #11. 10, 24, 36, 78, 85

PKI Public Key Infrastructure. 7, 23, 24, 57, 78

PnG Persona non-Grata. 16

PoC Proof of Concept. 3–5, 37, 63, 69, 71, 78, 79, 85

PSK Pre-Shared Key. 23

PUF Physically Unclonable Function. 12

RAM Random-Access Memory. 13, 43, 47, 69, 71

RE Requirement Enhancement. 20, 22, 72, 73

REE Rich Execution Environment. 49–51, 59, 60

RISC Reduced Instruction Set. 10, 84, 93

ROM Read-Only Memory. 13, 14, 46, 47, 57, 71, 72

ROS Rich OS. 14, 45, 51

RoT Root of Trust. 10, 14, 24, 35, 36, 39, 42, 45, 46, 57, 72, 73, 79

RPMB Replay-Protected Memory Block. 12, 50, 51, 60, 61, 63, 79, 82

RSU Road Side Unit. 2

SCA Side-Channel Attack. 8

SCFW System Controller Firmware. 47, 57, 78

SCU System Controller Unit. 46, 47

SDK Software Development Kit. 50, 51

SDL Security Development Lifecycle. 16

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

SECO Security Controller Firmware. 46, 47, 57

SHE Secure Hardware Extension. 34

sHSM Soft-Hardware Security Module. 43–45, 59, 63, 65, 66, 69, 71–73, 75, 78, 79, 84, 85

SL Security Level. 19, 20, 22, 72, 84

SoC System on Chip. 3, 4, 13, 14, 25, 29–37, 39, 41–44, 46, 47, 49, 72, 78–83, 85

SPD Secure Payload Dispatcher. 14

SPI Serial Peripheral Interface. 29, 30, 36

SPL Secondary Program Loader. 13, 48, 49, 55

STRIDE Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege. 16, 19, 20, 25

TA Trusted Application. 11, 14, 44, 45, 48–53, 56, 59, 60, 63, 65, 66, 75, 77, 80, 83–85

TCG Trusted Computing Group. 10, 30

TEE Trusted Execution Environment. xi, xiii, 9–12, 14, 32, 35, 36, 48, 71, 83

TLS Transport Layer Security. 23, 24

TOS Trusted OS. 14, 43

TPM Trusted Platform Module. 9, 10, 14, 15, 30, 35–37, 44, 83

TRA Threat and Risk Analysis. xi, xiii, 4, 5, 19, 24, 25, 69, 83, 84

UUU Universal Update Utility. 56, 57

V2X Vehicle-to-everything. xi, xiii, 1–3, 7, 19, 24, 29, 31, 85

VPN Virtual Private Network. 23

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[1] J. Pescatore, "Securing the "internet of things" survey," A SANS Analyst Survey, System Administration, Networking, and Security Institute (SANS), January 2014.

[2] J. Viega and H. Thompson, "The state of embedded-device security (spoiler alert: It's bad)," *IEEE Security Privacy*, vol. 10, pp. 68–70, Sep. 2012.

[3] H. Löhr, A.-R. Sadeghi, and M. Winandy, "Patterns for secure boot and secure storage in computer systems," in *2010 International Conference on Availability, Reliability and Security*, pp. 569–573, February 2010.

[4] M. Maidl, D. Kröselberg, J. Christ, and K. Beckers, "A comprehensive framework for security in engineering projects - based on iec 62443," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 42–47, Oct 2018.

[5] D. Tötzl, "C-ITS Einführung in Europa: Untersuchung von Alternativen zur Anbindung an eine V2X-Public Key Infrastructure," Master's thesis, UAS Technikum Wien, 2018. German.

[6] ETSI ES 202 663, "Intelligent Transport Systems (ITS); European profile standard for the physical and medium access control layer of Intelligent Transport Systems operating in the 5 GHz frequency band," Standard V1.1.0, European Telecommunications Standards Institute, Sophia Antipolis Cedex, France, January 2010.

[7] ETSI TS 102 940, "Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management," Technical Specification V1.3.1, European Telecommunications Standards Institute, Sophia Antipolis Cedex, France, April 2018.

[8] B. Lonc and P. Cincilla, "Cooperative its security framework: Standards and implementations progress in europe," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–6, June 2016.

[9] I. Ivanov, C. Maple, T. Watson, and S. Lee, "Cyber security standards and issues in v2x communications for internet of vehicles," in *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pp. 1–6, March 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[10] L. Gafencu and L. Scripcariu, "Security issues in the internet of vehicles," in *2018 International Conference on Communications (COMM)*, pp. 441–446, June 2018.

[11] European Commission, "Commission Delegated Regulation supplementing ITS Directive 2010/40/EU of the European Parliament and of the Council with regard to the provision of cooperative intelligent transport systems." `https://ec.europa.eu/info/law/better-regulation/initiatives/ares-2017-2592333_en`, [accessed April 2019], March 2019.

[12] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, pp. 145–152, July 2015.

[13] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper resistance mechanisms for secure embedded systems," in *17th International Conference on VLSI Design. Proceedings.*, pp. 605–611, Jan 2004.

[14] A. Fournaris, L. Pocero, and O. Koufopavlou, "Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks," *Electronics*, vol. 6, p. 52, 07 2017.

[15] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 361–372, June 2014.

[16] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[17] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[18] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad: Cross-privilege-boundary data sampling," in *CCS*, 2019.

[19] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[20] P. Kocher, R. Lee, G. Mcgraw, and A. Raghunathan, "Security as a new dimension in embedded system design," August 2004.

[21] D. D. Hwang, P. Schaumont, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security and Privacy*, vol. 4, pp. 40–49, Mar. 2006.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[22] P. Koopman, “Embedded system security,” *IEEE Computer*, pp. 95–97, July 2004.

[23] Cyber Security Research Alliance (CSRA), “Designed-In Cyber Security for Cyber-Physical Systems: Workshop Report by CSRA,” tech. rep., April 2013.

[24] B. Mccluskey, “Connected cars – the security challenge,” *Engineering Technology*, vol. 12, pp. 54–57, March 2017.

[25] L. Apvrille and Y. Roudier, “Towards the model-driven engineering of secure yet safe embedded systems,” in *Proceedings First International Workshop on Graphical Models for Security*, (Grenoble, France), pp. 15–30, April 2014.

[26] K. Markantonakis, D. K. Mayes;, K. E. Markantonakis, and K. E. Mayes, *Secure Smart Embedded Devices, Platforms and Applications*. New York, NY: New York, NY: Springer New York, 2014 ed., 2014.

[27] R. Soja, “Automotive security: From standards to implementation.” Freescale White Paper, January 2014.

[28] S. Cheruvu, A. Kumar, N. Smith, and D. M. Wheeler, *Demystifying Internet of Things Security: Successful IoT Device/Edge and Platform Security Deployment*. Berkely, CA, USA: Apress, 1st ed., 2019.

[29] R. Sanchez-Reillo, C. Sánchez Ávila, C. Lopez-Ongil, and L. Entrena, “Improving security in information technology using cryptographic hardware modules,” in *Proceedings. 36th Annual 2002 International Carnahan Conference on Security Technology*, pp. 120 – 123, February 2002.

[30] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. van Doorn, “A practical guide to trusted computing,” January 2008.

[31] L. Karter, L. Ferhati, I. Tafa, D. Saatciu, and J. Fejzaj, “Security evaluation of embedded hardware implementation,” in *2015 Science and Information Conference (SAI)*, pp. 1272–1276, July 2015.

[32] M. Wolf and A. Weimerskirch, “Hardware security modules for protecting embedded systems.” escrypt White Paper, 2013.

[33] S. Sau, J. Haj-Yahya, M. M. Wong, K. Yan Lam, and A. Chattopadhyay, “Survey of secure processors,” in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pp. 253–260, July 2017.

[34] M. Sabt, M. Achemlal, and A. Bouabdallah, “Trusted execution environment: What it is, and what it is not,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 57–64, August 2015.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[35] ARM Limited, Cambridge, England, *ARM Security Technology: Building a Secure System using TrustZone Technology*, April 2009. PRD29-GENC-009492CU.

[36] J. Winter, "Experimenting with arm trustzone – or: How i met friendly piece of trusted hardware," pp. 1161–1166, June 2012.

[37] X. Yan-ling, P. Wei, and Z. Xin-guo, "Design and implementation of secure embedded systems based on trustzone," in *2008 International Conference on Embedded Software and Systems*, pp. 136–141, July 2008.

[38] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pp. 445–451, November 2016.

[39] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Computing Surveys*, vol. 51, pp. 1–36, January 2019.

[40] B. Zhao, Y. Xiao, Y. Huang, and X. Cui, "A private user data protection mechanism in trustzone architecture based on identity authentication," *Tsinghua Science and Technology*, vol. 22, pp. 218–225, April 2017.

[41] GPD_SPE_010, "TEE Internal Core API Specification," Public Release Version v1.1.2, GlobalPlatform Device Technology, Reedwood City, CA, USA, November 2016.

[42] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-tee - an open virtual trusted execution environment," June 2015.

[43] ARM Limited, "Op-tee documentation." `https://optee.readthedocs.io/en/latest/`. Official Documentation.

[44] A. Nehal and P. Ahlawat, "Securing iot applications with op-tee from hardware level os," in *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 1441–1444, June 2019.

[45] C. Göttel, P. Felber, and V. Schiavoni, "Developing secure services for iot with op-tee: A first look at performance and usability," April 2019.

[46] O. Henniger, A. Ruddle, H. Seudié, B. Weyl, M. Wolf, and T. Wollinger, "Securing Vehicular On-Board IT Systems: The EVITA Project," in *25th Joint VDI/VW Automotive Security Conference*, (Ingolstadt, Germany), October 2009.

[47] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *14th International Conference on Information Security and Cryptology*, (Seoul, South Korea), November/December 2011.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

[48] JESD84-A441, “Embedded MultiMediaCard(e • MMC) e • MMC/Card Product Standard, High Capacity, including Reliable Write, Boot, Sleep Modes, Dual Data Rate, Multiple Partitions Supports, Security Enhancement, Background Operation and High Priority Interrupt (MMCA, 4.41),” JEDEC Standard 4.41, JEDEC Solid State Technology Association, March 2010.

[49] E. Zilberstein and A. Klein, “e.mmc security methods: A detailed overview of the different security methods one can use in an e.mmc storage device.” WesternDigital White Paper, July 2017.

[50] Anil Kumar Reddy, P. Paramasivam, and Prakash Babu Vemula, “Mobile secure data protection using emmc rpmb partition,” in *2015 International Conference on Computing and Network Communications (CoCoNet)*, pp. 946–950, December 2015.

[51] “Tpm-fail: TPM meets timing and lattice attacks,” in *29th USENIX Security Symposium (USENIX Security 20)*, (Boston, MA), USENIX Association, August 2020.

[52] K. Murdock, D. Oswald, F. D. Garcia, J. V. Bulck, D. Gruss, and F. Piessens, “Plundervolt: Software-based fault injection attacks against intel sgx,” in *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P’20)*, 2020.

[53] Y. Chen, Y. Zhang, Z. Wang, and T. Wei, “Downgrade attack on trustzone,” July 2017.

[54] L. Batina, P. Jauernig, N. Mentens, A.-R. Sadeghi, and E. Stapf, “In hardware we trust: Gains and pains of hardware-assisted security,” pp. 1–4, June 2019.

[55] ARM Limited and Contributors, “Trusted Firmware-A.” `https://trustedfirmware-a.readthedocs.io/en/latest/`, [accessed October 2019]. Official Documentation.

[56] ARM Limited, Cambridge, England, *System Hardware on ARM - Trusted Base System Architecture, Client*, 4th ed., October 2018.

[57] ARM Limited, Cambridge, England, *Trusted Board Boot Requirements CLIENT (TBBR-CLIENT) Armv8-A*, September 2018. DEN0006D, Beta 1.

[58] Y. Karim, J. Masters, G. Ben-Yossef, and P. Gerum, *Building Embedded Linux Systems*. O’Reilly Media, Inc., 2nd ed., 2008.

[59] T. Kai, X. Xin, and C. Guo, “The secure boot of embedded system based on mobile trusted module,” in *2012 Second International Conference on Intelligent System Design and Engineering Application*, pp. 1331–1334, January 2012.

[60] O. Khalid, C. Rolfes, and A. Ibing, “On implementing trusted boot for embedded systems,” in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 75–80, June 2013.

TU Wien Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[61] R. Rashmi and A. Karthikeyan, "Secure boot of embedded applications - a review," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 291–298, March 2018.

[62] G. Jin and L. Bo, "Design and implementation of a cryptographic file system for linux based on trusted computing platform," in *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, vol. 1, pp. 102–105, March 2011.

[63] W. Yu, W. Li, J. Wang, and C. Wei, "A Study of HSM Based Key Protection in Encryption File System," in *IEEE Conference on Communications and Network Security - Posters*, (Philadelphia, PA USA), October 2016.

[64] A. Shostack, *Threat modeling: Designing for Security*. Indianapolis, IN, USA: John Wiley and Sons, 2014.

[65] A. Moore, R. Ellison, and R. Linger, "Attack modeling for information security and survivability," Tech. Rep. CMU/SEI-2001-TN-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2001.

[66] S. P. Kadhirvelan and A. Söderberg-Rivkin, "Threat modelling and risk assessment within vehicular systems," master of science thesis in computer systems and networks, Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden, August 2014.

[67] A. Hadding and D. J. Zalewski, "Threat modeling in embedded systems." Florida Gulf Coast University, Summer 2012.

[68] N. Shevchenko, T. A. Chick, P. O'Riordan, T. P. Scanlon, and C. Woody, "Threat modeling: A summary of available methods," tech. rep., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, July 2018.

[69] N. Shevchenko, B. R. Frye, and C. Woody, "Threat modeling for cyber-physical system-of-systems: Methods evaluation," tech. rep., Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, September 2018.

[70] M. Howard and S. Lipner, *The Security Development Lifecycle*, vol. 34. June 2006.

[71] R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of microsoft's threat modeling technique," *Requirements Engineering*, vol. 20, June 2013.

[72] N. Mead, F. Shull, K. Vemuru, and O. Villadsen, "A hybrid threat modeling method," Tech. Rep. CMU/SEI-2018-TN-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2018.

[73] R. Khan, K. Mclaughlin, D. Laverty, and S. Sezer, "Stride-based threat modeling for cyber-physical systems," pp. 1–6, September 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[74] IEC/TS 62443-1-1, “Industrial communication networks – Network and system security – Part 1-1: Terminology, concepts and models,” Technical Specification Edition 1.0, International Electrotechnical Commission, Geneva, Switzerland, July 2009.

[75] IEC 62443-4-2, “Technical security requirements for IACS components,” International Standard Edition 1.0, International Electrotechnical Commission, Geneva, Switzerland, February 2019.

[76] J. Meier, A. Mackman, S. Vasireddy, M. Dunner, E. Ray, and A. Murukan, *Improving Web Application Security: Threats and Countermeasures.* Patterns & practices, Redmond, WA, USA: Microsoft Corporation, June 2003.

[77] Infineon Technologies, 81726 Munich, Germany, *Infineon's Security Solutions Portfolio*, June 2019. B180-I0041-V6-7600-EU-EC.

[78] Infineon Technologies, 81726 Munich, Germany, *OPTIGA TPM Application Note: Integration of an OPTIGA TPM SLx 9670 TPM 2.0 with SPI Interface in a Raspberry Pi 3 Linux environment with integrated TPM Driver*, 2019-03-14 ed., May 2019. Rev. 1.3.

[79] Infineon Technologies, 81726 Munich, Germany, *OPTIGA TPM Application Note: Integration of an OPTIGA TPM SLx 9670 TPM 2.0 with SPI Interface in a Raspberry Pi 4 Linux environment with integrated TPM Driver*, 2019-07-19 ed., July 2019. Rev. 1.0.

[80] Infineon Technologies, 81726 Munich, Germany, *Automotive application guide*, November 2018. B124-I0010-V3-7600-EU-EC-P.

[81] Infineon Technologies, 81726 Munich, Germany, *AURIXTM 32-bit microcontrollers for automotive and industrial applications*, February 2019. B158-I0090-V5-7600-EU-EC-P.

[82] NXP Semiconductors, *i.MX 6DualPlus/6QuadPlus Applications Processor Reference Manual*, July 2018. IMX6DQPRM, Rev. 2.

[83] NXP Semiconductors, *i.MX Linux Release Notes*, May 2019. IMXLXRN, Rev. L4.14.98-2.0.0_ga.

[84] NXP Semiconductors, *i.MX 8DualXPlus/8QuadXPlus Applications Processor Reference Manual*, November 2018. IMX8DQXPRM, Rev. D.

[85] NXP Semiconductors, *Security Reference Manual for i.MX 8DualXPlus/8QuadXPlus Application Processors*, May 2019. IMX8DQXPSRM, Rev. A.

[86] NXP Semiconductors, *i.MX 8QuadXPlus MEK Board Hardware User's Guide*, January 2019. IMX8QXPMEKHUG, Rev. 1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[87] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten, "ftpm: A software-only implementation of a TPM chip," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 841–856, USENIX Association, August 2016.

[88] NXP Semiconductors, *Secure Boot on i.MX 8 and i.MX 8X Families using AHAB*, May 2019. AN12312, Rev. 0.

[89] NXP Semiconductors and Contributors, "i.MX8/8x AHAB secure boot introduction." `https://source.codeaurora.org/external/imx/uboot-imx/tree/doc/imx/ahab/introduction_ahab.txt?h=imx_v2018.03_4.14.98_2.0.0_ga`. Rev. L4.14.98_2.0.0_ga.

[90] NXP Semiconductors and Contributors, "i.MX 8, i.MX 8X secure boot guide using AHAB." `https://source.codeaurora.org/external/imx/uboot-imx/tree/doc/imx/ahab/guides/mx8_mx8x_secure_boot.txt?h=imx_v2018.03_4.14.98_2.0.0_ga`. Rev. L4.14.98_2.0.0_ga.

[91] NXP Semiconductors and Contributors, "i.MX 8, i.MX 8X AHAB guide on SPL targets." `https://source.codeaurora.org/external/imx/uboot-imx/tree/doc/imx/ahab/guides/mx8_mx8x_spl_secure_boot.txt?h=imx_v2018.03_4.14.98_2.0.0_ga`. Rev. L4.14.98_2.0.0_ga.

[92] ARM Limited and Contributors, "Trusted Firmware-A." Rev. L4.14.98_2.0.0_ga.

[93] ARM Limited and Contributors, "Trusted Firmware-A user guide." `https://source.codeaurora.org/external/imx/imx-atf/tree/docs/user-guide.rst?h=imx_4.14.98_2.0.0_ga`. Rev. L4.14.98_2.0.0_ga.

[94] NXP Semiconductors, *i.MX Linux User's Guide*, April 2019. IMXLUG, Rev. L4.14.98-2.0.0_ga.

[95] NXP Semiconductors, *Code-Signing Tool User's Guide*, April 2019. CSTUG, Rev. 3.2.0.

TU Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.