D5.4
CARAMEL IDS/IPS Security Module

Topic	SU-ICT-2018
Project Title	Artificial Intelligence-based Cybersecurity for Connected and Automated Vehicles
Project Number	833611
Project Acronym	CARAMEL
Contractual Delivery Date	M19
Actual Delivery Date	M19
Contributing WP	WP5
Project Start Date	01/10/2019
Project Duration	30 Months
Dissemination Level	Public
Editor	DT-Sec
Contributors	DT-Sec

Document History		
Version	Date	Remarks
0.1	10/03/2021	Initial version, TOC
0.2	24/03/2021	Details on HSM integration added
0.3	07/04/2021	Details on NVidia Jetson AGX setup added EST demo described Missing texts added and structure completely filled Ready for internal review
0.9	20/04/2021	Pre-final version, ready for review
1.0	29/04/2021	Final version included internal review comments

DISCLAIMER OF WARRANTIES

This document has been prepared by CARAMEL project partners as an account of work carried out within the framework of the contract no 833611.

Neither Project Coordinator, nor any signatory party of CARAMEL Project Consortium Agreement, nor any person acting on behalf of any of them:

- makes any warranty or representation whatsoever, express or implied,
    - with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose, or
    - that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property, or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages, even if Project Coordinator or any representative of a signatory party of the CARAMEL Project Consortium Agreement, has been advised of the possibility of such damages) resulting from your selection or use of this document or any information, apparatus, method, process, or similar item disclosed in this document.

CARAMEL has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833611. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

DISCLOSURE STATEMENT

"The following document has been reviewed by the CARAMEL External Security Advisory Board as well as the Ethics and Data Management Committee of the project. Hereby, it is confirmed that it does not contain any sensitive security, ethical, or data privacy issues."

Table of Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial intelligence
API	Application Programming Interface
CA	Certificate authority
CAN	Controller area network
CCAM	Cryptographic Accelerator and Assurance Module
CPU	Central processing unit
D	Deliverable
EST	Enrolment over secure transport
HAB	High-assurance boot
HSM	Hardware security module
HTTP	Hypertext transfer protocol
HW	Hardware
I^2C	Inter-Integrated Circuit
ICT	Information and communication technologies
ID	Identifier
IF	Interface
IFD	Interface device
IP	Internet protocol
LTE	Long-term evolution
ML	Machine learning
MMU	Memory management unit
OSS	Open source software
PC	Personal computer
PCSC	Personal computer/smart card
PKCS	Public Key Cryptography Standards
RAM	Random access memory
RNG	Random number generator
ROM	Read-only memeory
RSA	Rivest, Shamir, & Adleman (public key encryption technology)
SCL	Serial clock line
SDA	Serial data
SSL	Secure Socket Layer
TCOS	Telekom Card Operating System
TEE	Trusted execution environment
ToC	Table of Contents
UC	Use Case
USB	Universal serial bus
V2X	Vehicle to X
WP	Work Package

Executive Summary

CARAMEL system components will address a wide range of security-related topics and technologies, from cyber threat detection (WP3), cyber-attack prevention (WP4), to in-depth defense mechanisms (WP5). The objective of WP5 is to provide the design, development, and prototype implementation of the CARAMEL anti-hacking device and in-depth defense solution. It will be based on different machine learning-based (ML) algorithms to detect and mitigate cyber-threats, while processing and collecting large volumes of data in future autonomous vehicle scenarios.

One of the possible attack vectors to V2X (vehicle to X or anything) infrastructure is the impersonation of V2X components such control units in the car, road-side infrastructure (e.g., traffic signs), or backend components, in order to steal sensitive data or interfere with the secure operation of transport infrastructure (e.g., to carry out terrorist attacks). These considerations are especially valid for the CARAMEL anti-hacking device. To counter the abovementioned attacks, trustworthy, unforgeable, and non-copyable identities must be established for the communication partners in a V2X setting. One way to achieve this goal is to integrate a hardware security module (HSM) into the embedded device that serves as a repository for private key data (for authentication and encryption purposes) as well as a cryptographic processor for sensitive operations. This task will address the hardware specification of the HSM as well as strategies for provisioning cryptographic keys and certificates into the HSM at manufacturing and later a deployment time.

This document describes the concrete and detailed configuration steps to integrate the CARAMEL hardware security module based on the Deutsche Telekom TCOS (Telekom Card Operating System) chip into three different anti-hacking device platforms chosen for the project. This work builds upon the specification laid out in D5.1 “Hardware Security Module Specifications”.

To this end we describe how the TCOS HSM module can be integrated using USB (universal serial bus card reader) and I2C (Inter-Integrated Circuit) interfaces on the physical layer. Furthermore, we show how to configure the anti-hacking device software stack, specifically the Crypto Service Container that provides high-level API (application programming interface) access to the HSM for applications deployed to the anti-hacking device in application Containers.

As a demonstration of the work done for this deliverable we have created a sample application that we describe in the form of a walkthrough. The goal of this sample application is to demonstrate the features we have implemented for the project as well as to provide a foundation for other partners to build their pillar-specific anti-hacking device applications that make use of the facilities provided by the TCOS HSM module.

An annex provides forward-looking information on the hardware and software features of the three anti-hacking device platforms that we currently support for the CARAMEL project. Building on this preliminary information we will continue to work on features like secure boot and secure firmware update as well as secure Docker container technology and describe these efforts in the forthcoming deliverables D5.5 and D5.6 of WP5.

1 Introduction

1.1 Project Overview

The rapidly growing connectivity of modern vehicles opens up numerous opportunities for new functions and attractive business models. At the same time, the potential for cyberattacks on vehicle networks is increasing. These attacks entail risks, especially with regard to functional safety and potential financial damage. CARAMEL's [1] goal is to proactively address modern vehicle cybersecurity challenges by applying advanced Artificial Intelligence (AI) and ML techniques, and also to continuously seek methods to mitigate associated safety risks. By adopting well-established methods from the ICT (Information and communication technologies) sector CARAMEL aims to develop an Anti-hacking IDS/IPS as a commercial product aimed towards the European automotive cyber security market and to demonstrate their value through comprehensive attack scenarios.

1.2 Document Scope

CARAMEL system components will address a wide range of security-related topics and technologies, from cyber threat detection (WP3), cyber-attack prevention (WP4), to in-depth defense mechanisms (WP5).

The objective of WP5 is to provide the design, development, and prototype implementation of the CARAMEL anti-hacking device and in-depth defense solution. It will be based on different machine learning-based algorithms to detect and mitigate cyber-threats, while processing and collecting large volumes of data in future autonomous vehicle scenarios. All these processes will be executed with state-of-art algorithms developed in WPs 3 and 4 of the CARAMEL project - updated in real time depending on the situational awareness about the underlying system at any time.

One of the possible attack vectors to V2X infrastructure is the impersonation of V2X components such control units in the car, road-side infrastructure (e.g., traffic signs), or backend components, to steal sensitive data or interfere with the secure operation of transport infrastructure (e.g., in order to carry out terrorist attacks). These considerations are especially valid for the CARAMEL anti-hacking device. To counter the abovementioned attacks, trustworthy, unforgeable, and non-copyable identities must be established for the communication partners in a V2X setting. One way to achieve this goal is to integrate a HSM into the embedded device that serves as a repository for private key data (for authentication and encryption purposes) as well as a cryptographic processor for sensitive operations.

The HSM has been specified in detail in deliverable D5.1 "Hardware Security Module Specifications" [2]. Based on this specification work this deliverable describes the actual implementation of the HSM integration on the hardware and software side. This integration is a prerequisite for the integration of use cases into the anti-hacking device that rely on the functionality of the integrated HSM.

This document will address the actual hardware integration steps of the HSM as well as strategies for provisioning cryptographic keys and certificates into the HSM at manufacturing and later a deployment time. The integration of the HSM will be demonstrated using an embedded hardware platform selected in the course of the project work.

1.3 Document Structure

This document is structured as follows:

Chapter 2 recaps the security features of the anti-hacking device and relates these efforts to other tasks in WP5.

Chapter 3 looks at the two integration options for the HSM: Either as a I2C-based module for direct connection to the I2C bus of embedded device, or as general-purpose USB-based solution that involves integration of the HSM using a USB card reader. The former will be demonstrated using the Coral Dev Board-based anti-hacking device, while the latter will be demonstrated using the NVidida Jetson AGX-based device.

In chapter 4 we will give a hand-on description of the software stack implemented on both anti-hacking device variants to support the integration of the anti-hacking device from the software side.

In chapter 5 we give details of the lab setup available for demonstrations and integration tests to be carried out by project partners responsible for the three pillars of CARAMEL. We provide an initial certification enrolment solution based on the Deutsche Telekom Trust Center as well as a simple application container using the Cryptographic Services Container that can be used a blueprint by project partners to integration their HSM-based solutions.

Note that this deliverable should be read in conjunction with D5.1 [2]. Unnecessary duplication of content has been avoided.

Additionally, some content planned for D5.5 “Secure Hardware Platform Specification” will be covered in the annex 7 in order to facilitate understanding some of the HSM integration decisions described in this document.

2 The CARAMEL Anti-hacking Device

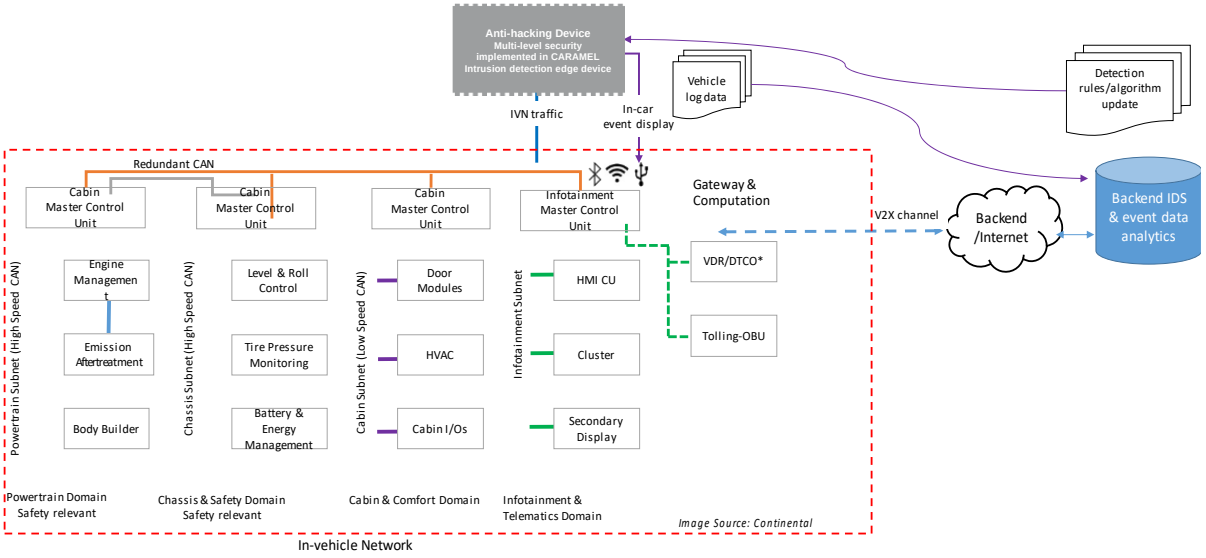

Figure 1: The anti-hacking device in the vehicle

The CARAMEL anti-hacking device is designed as a passive intrusion detection device that is integrated as an additional controller into the vehicle (see Figure 1). The anti-hacking device *passively* listens to the car's internal busses and systems, processes and aggregates raw data from sensors and communication controllers and uses machine learning (ML) and other heuristics to detect possible attacks against the vehicle's systems.

It then *actively* creates attack reports (events) and sends them to the CARAMEL backend. Details of the integration of the anti-hacking device into the different CARAMEL scenarios are described in the CARAMEL specification [4].

The anti-hacking device needs to be updated very frequently to run updated attack detection algorithms to counter newly discovered attack vectors. This requires frequent updates of the anti-hacking device firmware and application load. From a vehicle safety perspective any corruption of the anti-hacking device by malicious actors must be avoided at all costs.

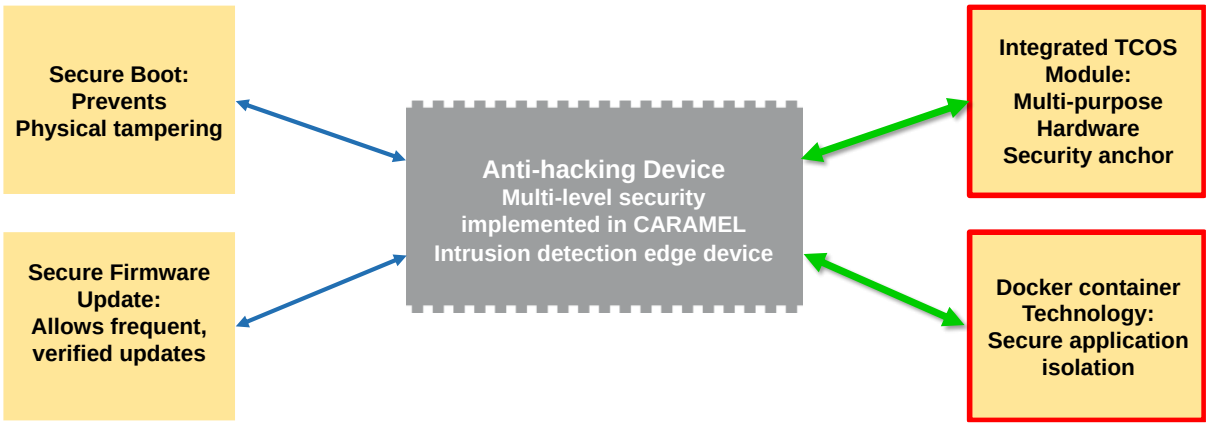

Figure 2: Anti-hacking device security features

To this end multiple security features will be implemented in the project to harden the anti-hacking software against any kind of attacks (see Figure 2):

- **Secure Boot:** The anti-hacking device hardware has fuses (write-once programmable storage locations) that contain the public keys of acceptable boot loader signatures. The anti-hacking

device only loads a correctly signed bootloader. The bootloader in turn verifies the signature of the Linux Kernel and only continues to load a verified kernel. These measures counter any physical tampering attacks on the boot medium.

- **Secure Firmware Update:** The anti-hacking device allows updating the firmware of the Internet (eg. over the vehicle's communication controller via LTE/5G). The anti-hacking device only accepts firmware update files that are properly signed by the anti-hacking device vendor. This protects the device against the installation of manipulated firmware images. In addition to this signature check the anti-hacking device implements also Secure Boot and would reboot to the last known safe state even if the secure firmware signature check were circumvented – effectively implementing multi-level security here.
- **Docker technology:** The anti-hacking device encapsulates the actual detection algorithms and also some system services into Docker containers. This has several advantages: It allows update of detection algorithms without a full firmware update. Additionally, the detection algorithms are separated by the protections offered by the Docker runtime against any mutual interference. As a last security measure, the anti-hacking device only accepts signed Docker images from pre-defined trusted sources, effectively also implementing multi-level security for Docker implementation on the anti-hacking device.
- **Integrated TCOS (HSM) module:** The anti-hacking device contains a hardware secure module (HSM) in the form of a Telekom Card Operating System (TCOS) security chip. Like a smartcard, the TCOS module offers secure storage of private key materials and certificates and the ability to run sensitive cryptographic operations securely on chip. The TCOS module offer these functionalities to Dockerized applications via a high-level security service also implemented as a Docker container.

This document builds upon the specification in D5.1 "Hardware Security Module Specifications" and describes in detail the integration steps of the TCOS module into three different anti-hacking device hardware instances as well as the actual configuration steps for the Docker cryptography service. The other security features shortly described above will be covered in more detail in forthcoming WP5 deliverables. Some preliminary information necessary for the understanding of the detailed instructions presented in this document will be introduced in an annex 7.

3 Integration of HSM into Anti-hacking Device

This section describes the hardware integration steps as well as some software configuration options in order to integrate the TCOS module into the three anti-hacking device platforms supported for the CARAMEL project at the time of this writing.

3.1 Integration via I2C into Coral Dev Board

Figure 3 shows how the TCOS module must be connected the I2C PIN headers on the Coral Dev Board:

Figure 3: TCOS module integration into Coral Dev Board

Table 1 gives the color coding of the cable for 4 I2C lines.

Colour	I2C line	Coral Dev Board PIN number	I2C device
Yellow	I2C2_SCL	5	/dev/i2c-1
Orange	Ground	6	
Red	3,3 V	1	
Brown	I2C2_SDA	3	/dev/i2c-1

Table 1: Colour coding and PIN assignments for I2C bus

The TCOS integration is performed using a IFD (interface device) handler library for the pcsc-lite daemon:

/usr/lib/pcsc/drivers/serial/libi2cifd-0.9.3.so

Additionally, a file libi2cifd must be installed into /etc/read.conf.d with the following contents:

```
# TCOS i2C Reader driver

FRIENDLYNAME      "TCOS i2C driver"
LIBPATH           /usr/lib/pcsc/drivers/serial/libi2cifd-0.9.3.so
CHANNELID         1
```

The IFD handler is configured in /etc/i2c_tcos.conf. For the Coral Dev Board this must contain the following:

```
# config file: /etc/i2c_tcos.conf
# for Version: 0.9.3

# i2C Device Address (7-bit)
DeviceAddress             = 72
#72
# Timeout after unsuccessful read (Microseconds - ms)
#ReadWrite_TimeOut    = 20000

# Number of retries after unsuccessful read
#ReadWrite_Retries      = 500

# Polling Timeout of the function IFD_POLLING_THREAD_WITH_TIMEOUT (ms)
Polling_TimeOut                   = 1000000

# Logfile of IFD Handler
Log                       = /tmp/i2c_tcos.log

# LogFile of i2C communication
I2CLog                    = /tmp/i2c_comm.log

# Device ID of i2c driver
DeviceID = /dev/i2c-1
```

Note: check that /dev/i2c-1 is included in the device list, otherwise the TCOS module will not be recognized.

3.2 Integration via USB into NVidia Jetson AGX

Figure 4: NVidia Jetson AGX with TCOS module attached via USB

Figure 4 shows a photo of the NVidia Jetson AGX device deployed in the DT-Sec ML lab in Berlin. For this photo the device was taken out of rack with all cables removed, but the SCM Microsystems, Inc. SCR331-LC1 / SCR3310 SmartCard reader attached to the one USB type A connector of the NVidia board. Inside the card reader we have deployed a TCOS smartcard in slot-in format (full-size cards are also available and will be used for demonstrations and tests in the project).

The SCM card reader is supported by the built-in driver pack of the pcsc-lite installation inside Crypto Service Container. Therefore, no additional configuration is necessary, only the USB port must be passed to the Crypto Service Container (see later sections).

The NVidia Jetson AGX device also has I2C PIN headers that can be used to attach the I2C-based TCOS module. [6]

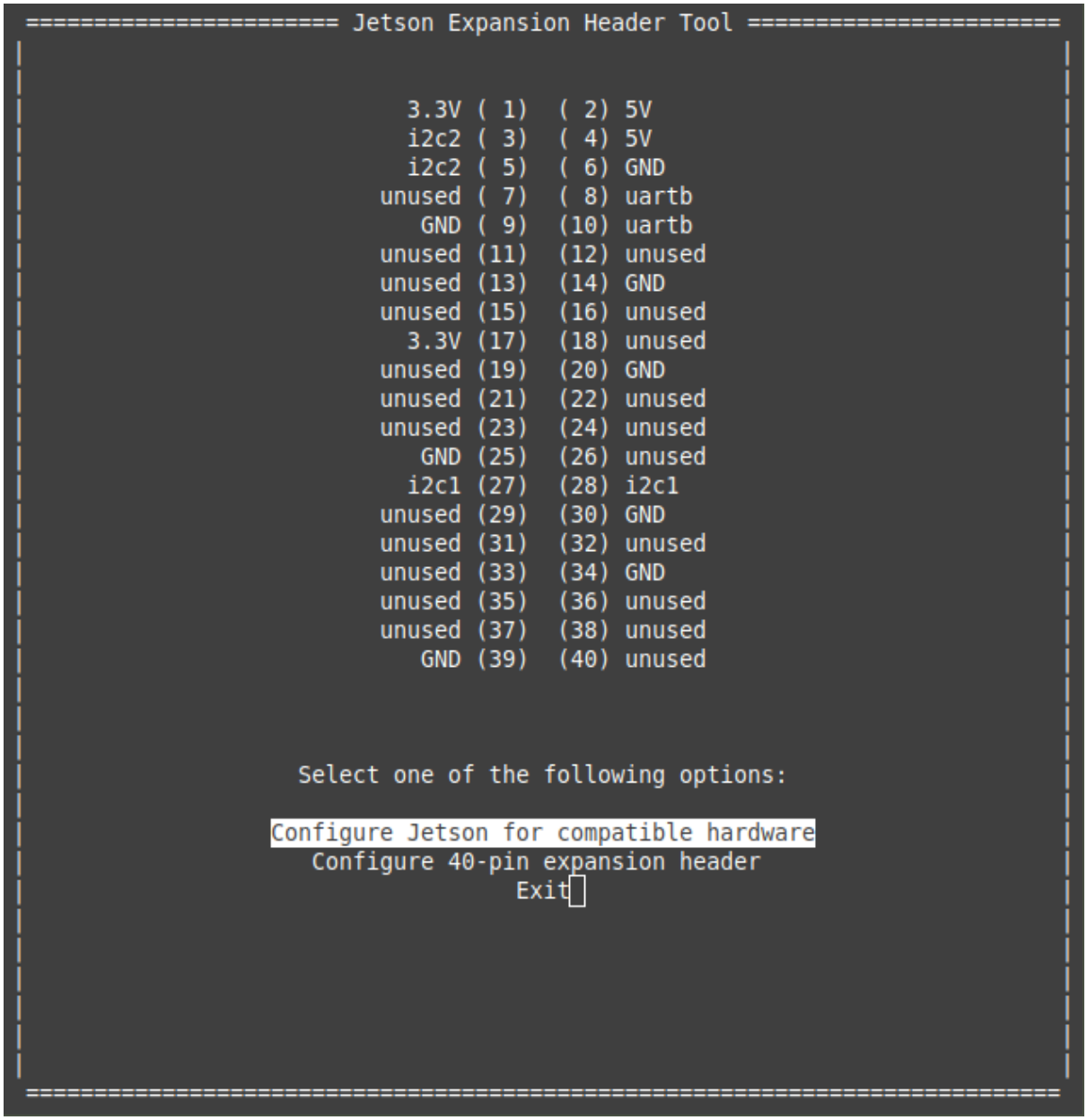

Figure 5: NVidia Jetson AGX PIN header assignment tool

Figure 5 gives an overview of the default assignments (screen shot of the PIN header assignment tool). The configuration files given in section 3.1 for the Coral Dev Board can be used without changes if the proper I2C headers are connected to the board. Since the NVidia Jetson AGX does not provide for clean integration of the cables and the module into the case we have pursued the USB dongle option to ease deployment of the solution.

3.3 Integration with Kontron K-Box K-BOX A-330 MX6

The Kontron K-Box A-330 MX6 iMXceet Dual S board [7] is another anti-hacking device platform that we support in the project. The advantage of the K-Box is its robust case that - combined with the wide-range power supply support (9 to 32 Volts) and integration CAN (Controller Area Network) bus support - make it ideal for integration into the vehicle. The K-Box can also be equipped with the Coral USB accelerator to support light machine learning tasks. Refer to section 7.4 for more information on the Kontron device we have procured for the CARAMEL project.

The K-Box supports the integration scenario with the USB dongle as described in section 3.2 for the NVidia Jetson AGX.

In addition to that, the K-Box has two physical I2C connections inside the case. See Figure 6 for an overview of the PIN header locations on the board [8].

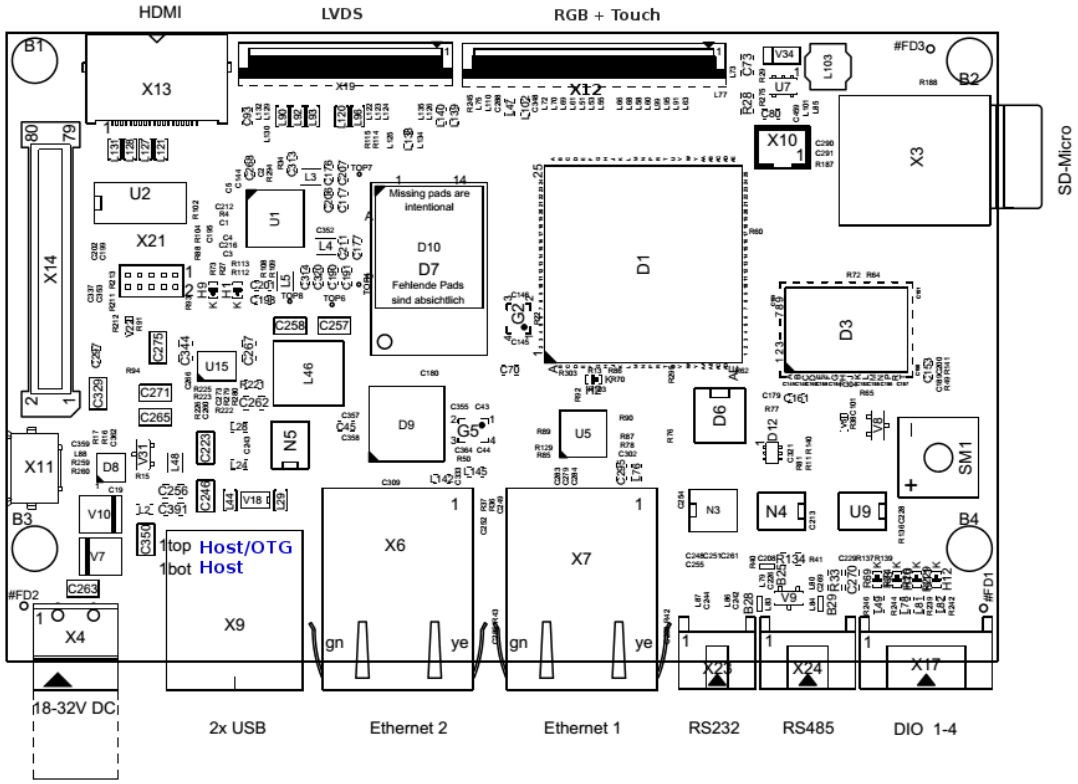

Figure 6: Kontron K-Box board layout and connector pinouts

Figure 7 and Figure 8 show where the I2C connections are located on the X12 and X19 connections. Figure 9 gives details on how the I2C connections are connected to the iMX6 SOC integrated on the board.

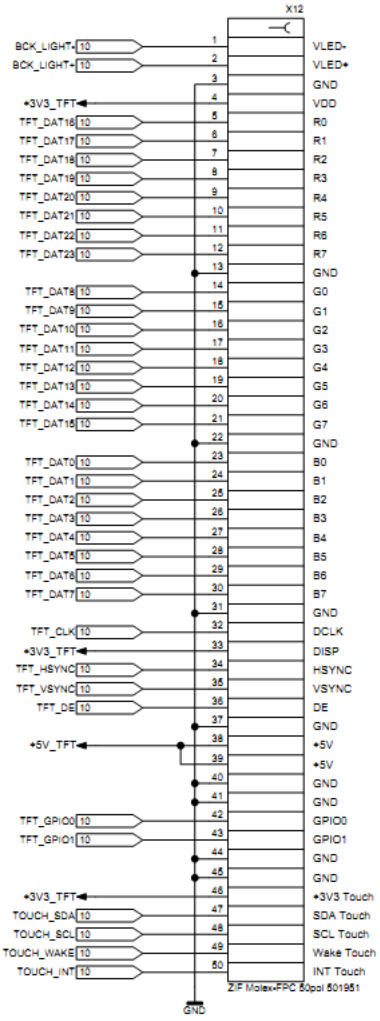

Figure 7: K-Box X12 connector pinout

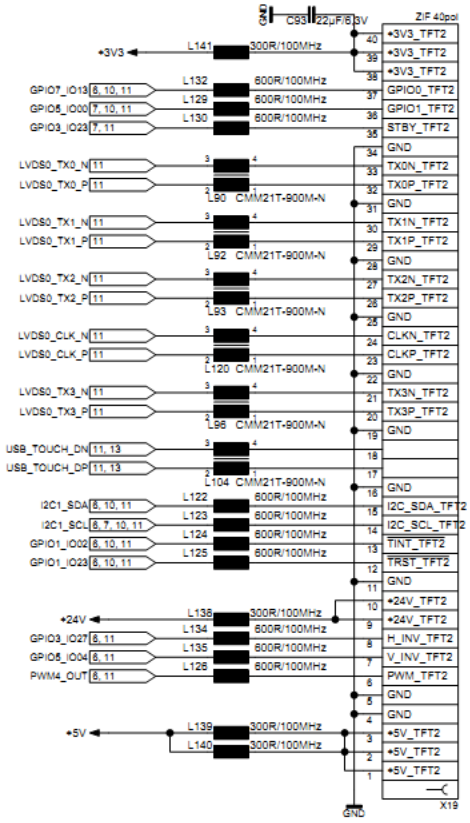

Figure 8: Kontron K-Box X19 connector pinout

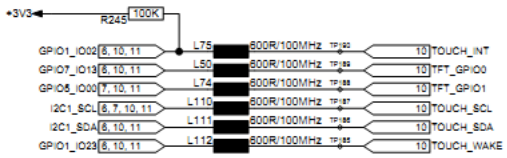

Figure 9: Kontron K-Box I2C SOC connection information

These diagrams give enough information to integrate the TCOS I2C module into the Kontron K-Box. At the time of this writing only the USB dongle integration method has been tested, however. We will pursue the I2C integration path in the course of the project.

4 Deployed Software Stack

4.1 Docker Image Software Stack

For security reasons a Crypto Service Docker Container is provided on the anti-hacking device that encapsulates all direct accesses to the TCOS module (see Figure 10). This encapsulation also enables easier access to the key material and cryptographic functions provided on the TCOS security module by applications.

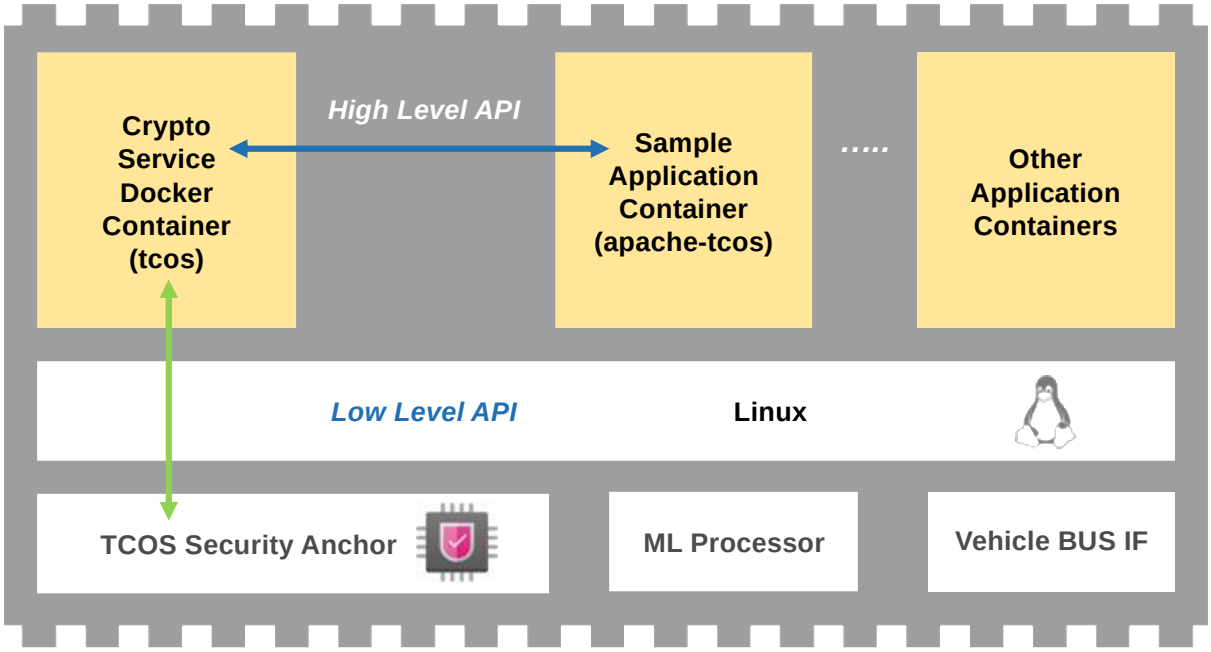

Figure 10: Anti-hacking software architecture

Application containers can make use of the cryptographic functions provided by the Crypto Service Container using a high-level API. For the demonstration setup described in detail in the following sections we provide a sample application container named “apache-tcos” that makes use of the crypto container for certificate provisioning. This container can be used a template by project partners to implement scenario-specific functionality that takes advantage of the integrated TCOS module.

The crypto service container totally hides from the application container(s) how the TCOS module is integrated into the system (I2C vs. USB integration).

4.1.1 Cryptographic Docker Service Container

The Docker image that directly accesses the TCOS module hardware can be downloaded from here:

https://caramelx.mine.bz/caramelx-h2020-prv_dcs/coral/tcos.tar

On the Coral Dev Board, install the Docker image like this:

docker load --input tcos.tar

4.1.2 High-level APIs / Client Access from Application Containers

We provide a template / demo docker container that project partners can use and expand for their own applications. This can be downloaded from:

https://caramelx.mine.bz/caramelx-h2020-prv_dcs/coral/apache-tcos.tar

On the anti-hacking device, install the Docker image like this:

docker load --input apache-tcos.tar

A full docker-compose file and some accompanying scripts can be downloaded from here:

https://caramelx.mine.bz/caramelx-h2020-prv_dcs/coral/apache-demo.tar

4.1.3 Detailed Setup of the Provided Docker Images

The docker-compose.yml file must be changed in order to reflect the device setup (change highlighted device mappings appropriately) on the anti-hacking device:

```
version: '3'

services:
  tcos:
    container_name: tcos-1
    image: plasma.atriumberlin.de/tcos
    devices:
      - "/dev/i2c-0:/dev/i2c-0"
      - "/dev/i2c-1:/dev/i2c-1"
      - "/dev/i2c-2:/dev/i2c-2"
      - "/dev/bus/usb/001/001:/dev/bus/usb/001/001"
      - "/dev/bus/usb/001/002:/dev/bus/usb/001/002"
      - "/dev/bus/usb/001/003:/dev/bus/usb/001/003"
    restart: on-failure
    environment:
      - tokenUrl=pkcs11:model=TCOS%203.0%20NetKey;manufacturer=T-Systems%20International%20GmbH;serial=17360000442334;token=8949017360000442334
    volumes:
      - p11-server:/var/p11-server

  apache-tcos:
    container_name: apache-tcos-1
    image: plasma.atriumberlin.de/apache-tcos
    restart: on-failure
    volumes:
      - p11-server:/var/p11-server
      - /home/root/apache-demo/html:/var/www/html/status
    ports:
      - "8080:80"
      - "8443:443"
    depends_on:
      - tcos

volumes:
  p11-server:
```

Additionally, you must adapt the tokenUrl (highlighted) in order to access the specific TCOS module you have connected via I2C or USB. Specifically, the serial and token values needed to be determined. In order to do this, launch all containers via

docker-compose up

first. Then exec into the tcos-1 container:

There, display the required information from the installed TCOS module:

pkcs11-tool --module /usr/lib/pkcs11/libpkcs11tcos3NetKey_ARM-PCSC-1.8.2.so -L

Eg. the output from our test module is:

```
Available slots:
Slot 0 (0x1000): TCOS i2C driver 00 00
  token label        : 8949017360003147278
  token manufacturer : T-Systems International GmbH
  token model        : TCOS 3.0 NetKey
  token flags        : login required, rng, SO PIN locked, SO PIN to be changed, token initialized, user PIN
to be changed
  hardware version   : 3.2
  firmware version   : 0.0
  serial num         : 17360003147278
  pin min/max        : 6/24
```

Take the serial number 17360003147278 and the token label 8949017360003147278 and change the line in the docker-compose.yml file accordingly:

```
    - tokenUrl=pkcs11:model=TCOS%203.0%20NetKey;manufacturer=T-
Systems%20International%20GmbH;serial=17360003147278;token=8949017360003147278
```

Now restart the containers (docker-compose down followed by docker-compose up) and exec again into the tcos-1 container.

You now have to change the PIN from the initial zero PIN to “123456”. Always use “123456” in a test environment, otherwise scripts and demos assuming this specific PIN might render the TCOS module unusable if another PIN is set.

Issue the following command in the tcos-1 container to change the initial PIN to “123456”:

```
pkcs11-tool --module /usr/lib/pkcs11/libpkcs11tcos3NetKey_ARM-PCSC-1.8.2.so --init-pin
```

Be careful to enter the PIN “123456” here. You can now list the cryptographic objects on the module like this:

```
pkcs11-tool --module /usr/lib/pkcs11/libpkcs11tcos3NetKey_ARM-PCSC-1.8.2.so -login -p 123456 -O
-M
```

The connection between cryptographic container (tcos-1) and application container(s) (in the example/demo apache-tcos-1) is done using the remote access layer described in deliverable D5.1 [2]. In order to test the connection exec into the apache-tcos-1 container:

```
docker exec -it apache-tcos-1 bash
```

Here, issue the following command to list the cryptographic objects, but now using the remote access library:

```
pkcs11-tool --module /usr/lib/pkcs11/p11-kit-client.so -login -p 123456 -O -M
```

This command should basically yield the same results as directly in the tcos-1 cryptographic container.

5 Demonstration description

In this section we describe a demo setup that is intended to highlight:

- the high-level usage of the TCOS crypto container from another, application-level crypto container and
- one possible way to procure securely and automatically certificates from a trust center, in this case the Deutsche Telekom Auto-enrolment trust center.

To this end we have create a simple demo container called “apache-tcos” that contains a sample application (in this case the Apache HTTP server with PHP enabled) that procures a new certificate for a private key on the TCOS module upon first startup. The certificate is then delivered via the integrated HTTP server. It is expected that project partners use this demo container as a blueprint to build more complex applications and certificate provisioning scenarios for CARAMEL pillars.

5.1 Concrete demonstration setup

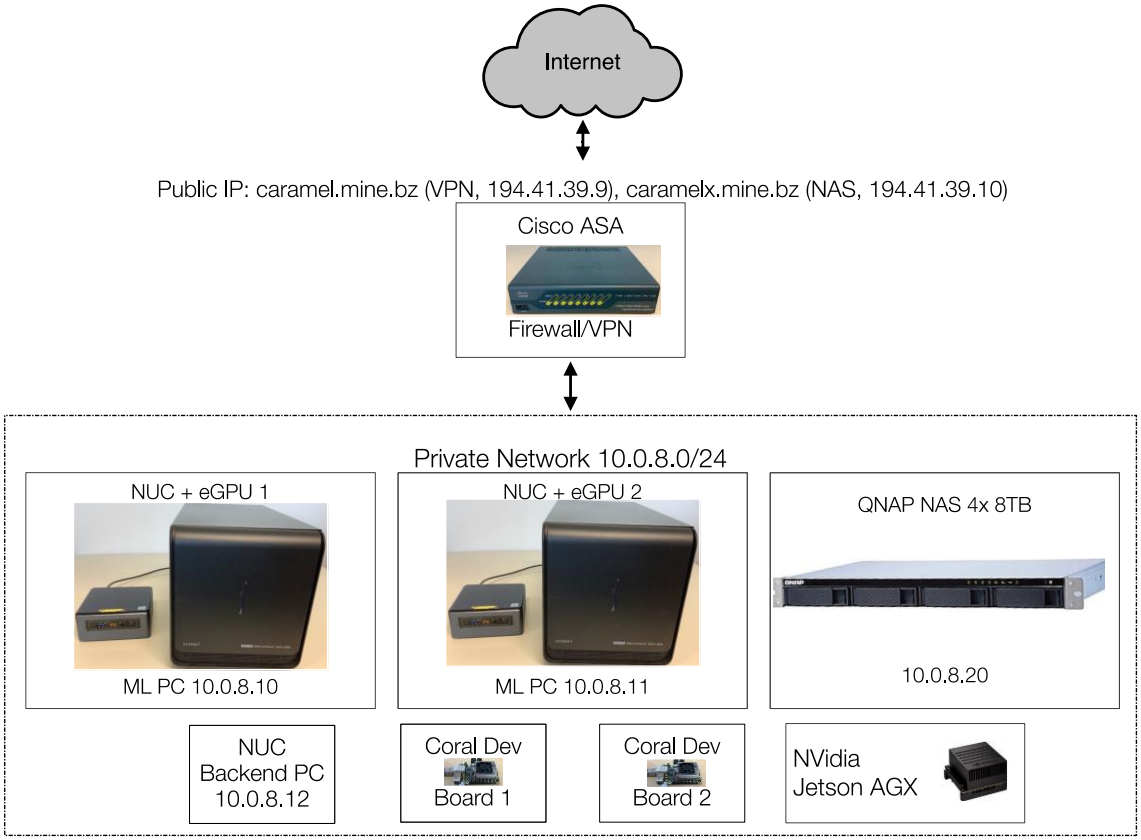

Figure 11: DT-Sec ML lab in Berlin

Figure 11 gives an overview of the ML lab we set up in Berlin for the CARAMEL project.

Note: A Kontron K-Box will be added to the lab setup at a later stage after the publication of this document.

Name	IP address	DNS name	Description
	10.0.8.1	caramel.mine.bz	Cisco ASA Firewall for Cisco Anyconnect VPN access to the lab infrastructure

caramel1	10.0.8.10		NUC PC with attached NVidia eGPU for machine learning and CARLA simulation purposes 16 GB RAM Core i7 processor 32 GB boot SSD (Optane memory) 2 TB data and home/docker disk eGPU with NVidia GTX 1080 Ti (11GB GPU RAM) Ubuntu 18.04
caramel2	10.0.8.11		NUC PC with attached NVidia eGPU for machine learning and CARLA simulation purposes 32 GB RAM Core i7 processor 1TB boot SSD 2 TB data and home/docker disk eGPU with NVidia RTX 2070 (8GB GPU RAM) Ubuntu 18.04 Coral USB Accelerator attached (Update: currently removed for tests with the Coral Dev Board)
caramel3	10.0.8.12		NUC PC (NUC7i7BNB) with attached NVidia eGPU for backend containers 16 GB RAM Core i7 processor 256 TB data and home/docker disk Intel® Iris® Plus Graphics 650 (processor graphics) Ubuntu 18.04
caramel-nas	10.0.8.20 194.41.39.10	caramelx.mine.bz	QNAP NAS device 8 GB RAM 20 TB HDD (4 x 8 TB RAID5)
kvm-ip	10.0.8.30		KVM-IP switch with VNC protocol access Currently attached to caramel-jetson
coral1	10.0.8.50		Coral Dev Board 1 GB RAM 32 GB SD card
coral2	10.0.8.51		Coral Dev Board 1 GB RAM

			32 GB SD card
caramel-jetson	10.0.8.55		NVidia Jetson AGX 32 GB RAM 1 TB SSD

Table 2: Computing resources deployed in the ML lab in Berlin

Table 2 gives an overview of the devices deployed in the lab. Of special interest are the two Coral Dev boards (coral1 and coral2) as well as the NVidia Jetson AGX device deployed in the lab. These can be used by project partners remotely. The following remote access options exist:

- Cisco AnyConnect access to caramel.mine.bz (194.41.39.9) for project partners with individual user ID/passwords
- SSH access (over VPN) to coral1 and coral2 with root and no password
- SSH access (over VPN) to caramel-jetson with user ID and password (same as for VPN access)
- VNC access to kvm-ip (over VPN) to caramel-jetson, VNC port 5900 with user ID and password (same as for VPN access)

This facility allows partners to develop application containers for their pillar-specific scenarios to test their machine learning algorithms on the target platforms including usage of the attached TCOS-based HSMs without the need for any physical access to this hardware.

5.2 Demonstration Walk-through

We have prepared a sequence of demonstration steps to showcase the different elements of the HSM integration we have implemented for this deliverable. The steps are as follows:

5.2.1 Login via VPN

Use a Cisco AnyConnect client for your platform to securely access the lab infrastructure.

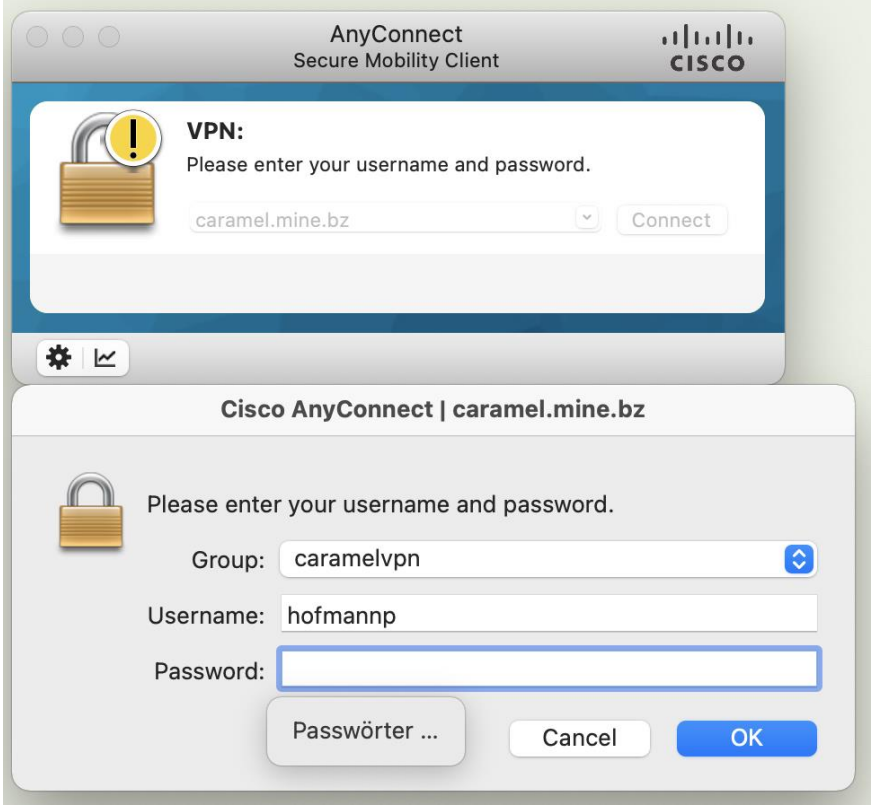

Figure 12: Cisco AnyConnect login window

Enter Username and Password in the Cisco AnyConnect login screen (see Figure 12). After the VPN session is connected, the VPN allows access to the 10.0.8.0/24 subnet of the lab network.

5.2.2 SSH to one of the anti-hacking devices (coral1, coral2, or caramel-jetson)

Use the following commands to access the command line of the anti-hacking devices:

- coral1: ssh root@10.0.8.50 (no password)
- coral2: ssh root@10.8.8.51 (no password)
- caramel-jetson: ssh username@10.0.8.55 (password from VPN access)

On the Coral Dev Board devices you will find our CARAMEL-specific Yocto firmware load with limited command line binaries available. The caramel-jetson device, however, is based (at the time of this writing) on Ubuntu 18.04 and supports a large number of commands for test and development purposes.

5.2.3 Start of Demo Containers

The following assumes that the apache-demo.tar archive is unpacked under the src directory (refer to section 4.1.3 for the download link to this demo archive).

Here we reproduce a trace of the first startup of the tcos and apache-tcos containers. The apache-tcos log contains verbose information from the EST (enrolment over secure transport) provisioning tool that requests the initial certificate from our Deutsche Telekom auto-enrolment trust center:

```
caramel-jetson:~/src/apache-demo$ docker-compose up
Creating network "apachedemo_default" with the default driver
Creating volume "apachedemo_p11-server" with default driver
Creating tcos-1 ...
Creating tcos-1 ... done
Creating apache-tcos-1 ...
```

```
Creating apache-tcos-1 ... done
Attaching to tcos-1, apache-tcos-1
tcos-1        | P11_KIT_SERVER_ADDRESS=unix:path=/var/p11-server/p11-server.sock; export
P11_KIT_SERVER_ADDRESS;
tcos-1        | P11_KIT_SERVER_PID=17; export P11_KIT_SERVER_PID;
apache-tcos-1 | AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set
the 'ServerName' directive globally to suppress this message
tcos-1        | Successfully changed to user polkitd
tcos-1        | 13:11:41.260: Loading rules from directory /etc/polkit-1/rules.d
tcos-1        | 13:11:41.260: Loading rules from directory /usr/share/polkit-1/rules.d
tcos-1        | 13:11:41.262: Finished loading, compiling and executing 3 rules
tcos-1        | Entering main event loop
tcos-1        | Connected to the system bus
tcos-1        | 13:11:41.265: Acquired the name org.freedesktop.PolicyKit1 on the system bus
apache-tcos-1 | Using slot 0 with a present token (0x10)
apache-tcos-1 | Using slot 0 with a present token (0x10)
apache-tcos-1 | ..................................+++++
apache-tcos-1 | ..................................................+++++
apache-tcos-1 | writing RSA key
apache-tcos-1 | Using slot 0 with a present token (0x10)
apache-tcos-1 | Created public key:
apache-tcos-1 | Public Key Object; RSA 2048 bits
apache-tcos-1 |   label:     Public Import Key
apache-tcos-1 |   ID:        df024e01
apache-tcos-1 |   Usage:     encrypt, verify
apache-tcos-1 | Using slot 0 with a present token (0x10)
apache-tcos-1 | Created private key:
apache-tcos-1 | Private Key Object; RSA
apache-tcos-1 |   label:     Private Import Key
apache-tcos-1 |   ID:        df024e01
apache-tcos-1 |   Usage:     decrypt, sign
apache-tcos-1 | [ID-STORAGE]          failed to open certificate file
apache-tcos-1 | [ID-STORAGE]          failed to process certificate file
apache-tcos-1 | [CEST]          failed to load identity from filesystem
apache-tcos-1 | [CEST]          found no valid identity to restore
apache-tcos-1 | [CEST-CLI]    will append intermediate CA certificates when storing leaf certificate
apache-tcos-1 | [ID-STORAGE]          trying to load certificate from: /root/.est/id_rsa.cert.pem
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connecting to autocert.test.telesec.de on port 443
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connection established
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       performing tls-handshake
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       server certificate valid
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       using cipher suite TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       written ssl data: 243/243
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       retrieved 2684 of max 4000 bytes
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       completed https transaction
apache-tcos-1 | [EST-HTTPS]  https certificate request succesfull
apache-tcos-1 | [ID-ENCODING]         found valid certificate
apache-tcos-1 | [ID-ENCODING]         found valid certificate
apache-tcos-1 | [ID-STORAGE]          prepared single certificate for writing
apache-tcos-1 | [ID-STORAGE]          succesfully wrote certificate to file
apache-tcos-1 | [ID-STORAGE]          succesfully wrote certificate to file
apache-tcos-1 | [CEST]          stored trust chain
apache-tcos-1 | [ID-STORAGE]          failed to open certificate file
apache-tcos-1 | [ID-STORAGE]          failed to process certificate file
apache-tcos-1 | [CEST]          failed to load identity from filesystem
apache-tcos-1 | [CEST]          found no valid identity to restore
apache-tcos-1 | [CEST-CLI]    using 'sdq22oa07ea2h455N' for authentication
apache-tcos-1 | [CEST-CLI]    identity will be identified by 'CN=secure_gw,OU=Clients,O=Bosch'
apache-tcos-1 | [ID-STORAGE]          trying to load certificate from: /root/.est/id_rsa.cert.pem
apache-tcos-1 | [CEST]          generating new private key
apache-tcos-1 | [IDENTITY-SOFTWARE]       generating new private key now. This may take a while...
apache-tcos-1 | [IDENTITY-SOFTWARE]       finished generation of new private key
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connecting to autocert.test.telesec.de on port 443
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connection established
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       performing tls-handshake
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       server certificate valid
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       using cipher suite TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       written ssl data: 243/243
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       retrieved 2684 of max 4000 bytes
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       completed https transaction
apache-tcos-1 | [EST-HTTPS]  https certificate request succesfull
apache-tcos-1 | [ID-ENCODING]         found valid certificate
apache-tcos-1 | [ID-ENCODING]         found valid certificate
apache-tcos-1 | [EST-PKCS10]          signing the csr
```

```
apache-tcos-1 | [EST-PKCS10]          pem-encoding the csr
apache-tcos-1 | [EST-PKCS10]          generated csr successfully
apache-tcos-1 | [EST-PKCS10]          generated csr successfully
apache-tcos-1 | [EST]          generated csr:
apache-tcos-1 |
MIICnTCCAYUCAQAwNjEOMAwGA1UECgwFQm9zY2gxEDAOBgNVBAsMB0NsaWVudHMxEjAQBgNVBAMM
apache-tcos-1 | CXNlY3VyZV9ndzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN/TTiHu6u+ermbwAAIb
apache-tcos-1 | BXoqV+xPBjibY+qHEGKDFdyCijLSi3f5Kk8gJ7e4T4ygj7gyk0gpRiz7lKzQyzyyHtu2T0/5+lzs
apache-tcos-1 | /P7j3fJriTIEz/yi/kTE2tmTxRzXUt5xE+w4o7gNqrxxFCHZCeKU4f8Hb13a1ou0NBb4GgKW8AIY
apache-tcos-1 | XvpfPhg6pUrcaPAQoN5t8BJHsfF7nkWi7Z5tKfa30sEjYVUcc3gT5MV08sSOQBOlnDw9L/HcaSoS
apache-tcos-1 | NEn3yitry4dju7u0Fza0EXrGE1HENJbbDOexTbpklMOM84EdN466jc5r+GZeCBkN/rI9Bd9vGUIo
apache-tcos-1 | 3qXpqsoKKo2oFqnpFpkCAwEAAaAiMCAGCSqGSIb3DQEJBzETDBFzZHEyMm9hMDdlYTJoNDU1TjAN
apache-tcos-1 | BgkqhkiG9w0BAQsFAAOCAQEAo42xq+xoHU7S5fXmlGNvZW9ogKhWlCQKrBHyzJt6nHltgrgSEU6g
apache-tcos-1 | PsdpddgwaUJiWpA05OBR/CZZd0erNwljIyHJ9VFnhh0fVyU0YT3040VDlt0fx7q+iQn853yW37wP
apache-tcos-1 | cgQP5++tzjA9PTNUnzqstEpOpiSmPkM9BdbVTv2pSFQYFzZ8Pg6zVmuc1eLHrxtAdumRysNy2DMS
apache-tcos-1 | k8b/FT97JCfHgbfH96Cefru8ZmwztgpMlyIudgv91UlVwDz8Qhl3/GICp/pPd56OJfN6aWbr9++t
apache-tcos-1 | 0BB2cgCze4cFkRYWsmYaAxdNwSLCuD/OQSJ7G5/+igbGJt/+8ERL5V10yWOhbg==
apache-tcos-1 |
apache-tcos-1 |
apache-tcos-1 | [EST]          attempting enrollment for 'CN=secure_gw,OU=Clients,O=Bosch' with challenge
'sdq22oa07ea2h455N'
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connecting to autocert.test.telesec.de on port 443
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       connection established
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       performing tls-handshake
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       server certificate valid
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       using cipher suite TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       written ssl data: 1181/1181
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       retrieved 1384 of max 4000 bytes
apache-tcos-1 | [HTTPS-NET-MBEDTLS]       completed https transaction
apache-tcos-1 | [EST-HTTPS]  https certificate request succesfull
apache-tcos-1 | [EST]          received response:
apache-tcos-1 |
MIAGCSqGSIb3DQEHAqCAMIACAQExADCABgkqhkiG9w0BBwEAAKCAMIID2TCCAsGgAwIBAgIBGzANBgkqhkiG9w0BAQs
FADAfMR0wGwYDVQQDExRUZXN0IEJvc2NoIENBIENsaWVudDAeFw0yMTA0MDYxMzEyMzFaFw0yMjA0MDYyMzU5NTla
MDYxDjAMBgNVBAoTBUJvc2NoMRAwDgYDVQQLEwdDbGllbnRzMRIwEAYDVQQDDAlzZWN1cmVfZ3cwggEiMA0GCSqGS
Ib3DQEBAQUAA4IBDwAwggEKAoIBAQDf004h7urvnq5m8AACGwV6KlfsTwY4m2PqhxBigxXcgooy0ot3+SpPICe3uE+MoI+4M
pNIKUYs+5Ss0Ms8sh7btk9P+fpc7Pz+493ya4kyBM/8ov5ExNrZk8Uc11LecRPsOKO4Daq8cRQh2QnilOH/B29d2taLtDQW+Bo
ClvACGF76Xz4YOqVK3GjwEKDebfASR7Hxe55Fou2ebSn2t9LBI2FVHHN4E+TFdPLEjkATpZw8PS/x3GkqEjRJ98ora8uHY7u7
tBc2tBF6xhNRxDSW2wznsU26ZJTDjPOBHTeOuo3Oa/hmXggZDf6yPQXfbxlCKN6l6arKCiqNqBap6RaZAgMBAAGjggEHMIIB
AzAfBgNVHSMEGDAWgBTBeEVOLnfXCS1fmYQFEZFBA/L2OzAdBgNVHQ4EFgQU8VxpAoapiqYVSHc/+04wnJsZAzIwDgY
DVR0PAQH/BAQDAgWgMBMGA1UdJQQMMAoGCCsGAQUFBwMCMAkGA1UdEwQCMAAwTQYDVR0fBEYwRDBCoECgP
oY8aHR0cDovL2NybC5hZWNhLnRlc3QudGVsZXNlYy5kZS9jcmwvVGVzdF9Cb3NjaF9DQV9DbGllbnQuY3JsMEIGCCsGAQU
FBwEBBDYwNDAyBggrBgEFBQcwAYYmaHR0cDovL29jc3AuYWVjYS50ZXN0LnRlbGVzZWMuZGUvb2NzcHIwDQYJKoZIhvc
NAQELBQADggEBACqqMRxYi02V6zhs2725EF+njIFxVtTGJMtRlFi4CY3Tcw/bbmxGYGFXP4pJDW+AKvWSQOfnej/WslyjtzsE
xvmZGkQ4GEVLnbk931c9zg5NiDGd9X+5I5WTn9Epp1o3bW0dnSsvM119W+Rqr579tiM/U+eInl6Ao/jQsgjA7cN90v46fZ5CzAD
jG1NS2weB6hHWuAVLYRVTYfIQR0LQKFxUCSUS2jk2gPbgGFiyYJKeTmlAGKLUDlCxSscKKkzGt936gLRIL+bpEsg75zXQjAp
tTvKtI/sQcwmUYLJt2tlwB/0Dz3w7cn7mKDPjgvRskEOAzgpo5PDKnp1rmVEeU9wAADEAAAAAAAAA
apache-tcos-1 | [ID-STORAGE]          prepared single certificate for writing
apache-tcos-1 | [ID-STORAGE]          succesfully wrote certificate to file
apache-tcos-1 | [ID-STORAGE]          storing single certificate succesfull
apache-tcos-1 | [ID-STORAGE]          succesfully wrote private key to file
apache-tcos-1 | [ID-STORAGE]          storing private key succesfull
apache-tcos-1 | [CEST]          sucessfully generated new identity
```

After this initial configuration is performed, the TCOS module is initialized with the new certificate from the Deutsche Telekom auto-enrolment trust center. We have implemented a simple web interface to show details from this certificate (Figure 13):

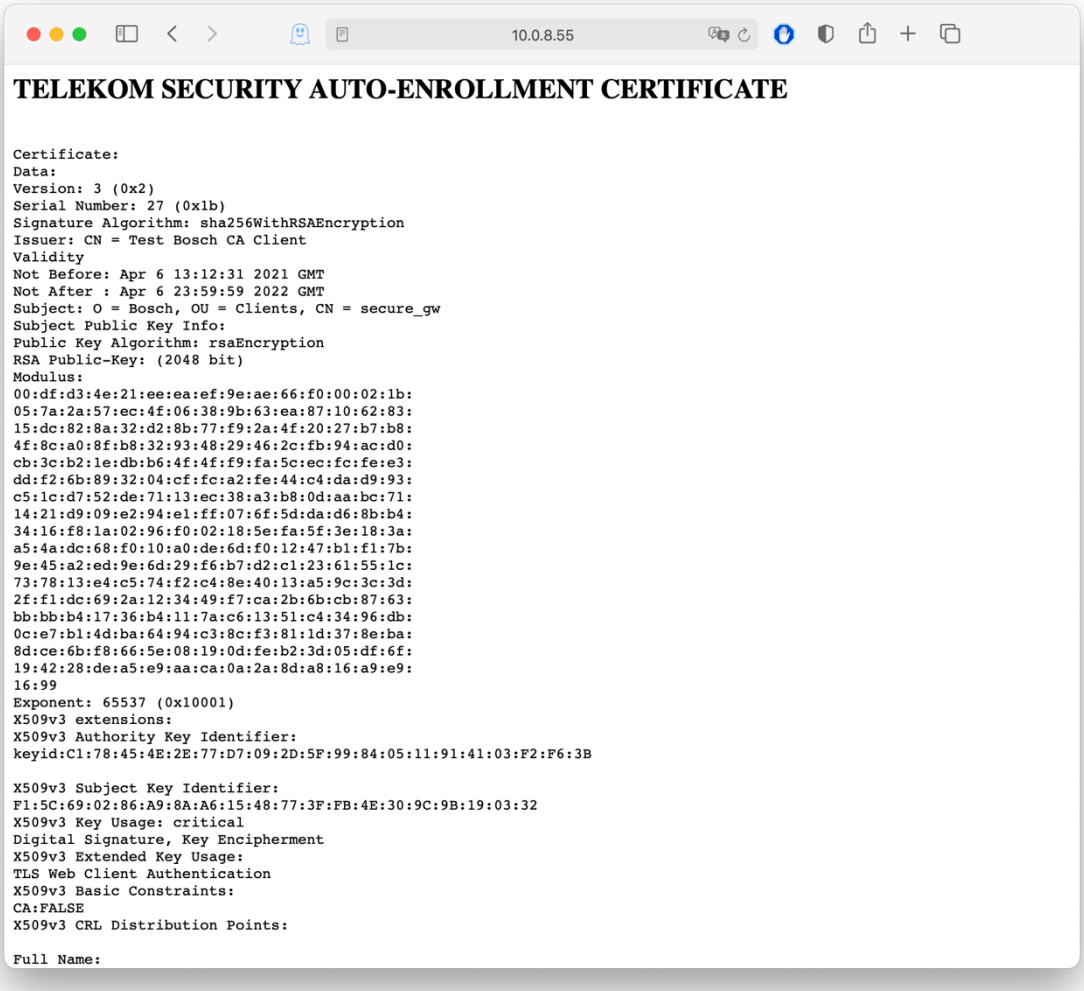

Figure 13: Sample Web Application

6 Conclusions and Next Steps

This specification document has described the concrete and detailed configuration steps to integrate the CARAMEL hardware security module based on the Deutsche Telekom TCOS chip into three different anti-hacking device platforms chosen for the project.

In chapter 3 we have described how the TCOS HSM module can be integrated using USB (card reader) and I2C interfaces on the physical layer.

In chapter 4 we have described how to configure the anti-hacking device software stack, specifically the Crypto Service Container that provides high-level API access to the HSM for applications deployed to the anti-hacking device in application Containers.

In chapter 5 we have described the sample application in the form of a walkthrough. The goal of this sample application is to demonstrate the features we have implemented for the project as well as to provide a foundation for other partners to build their pillar-specific anti-hacking device applications that make use of the facilities provided by the TCOS HSM module.

In annex 7 we have provided forward-looking information on the hardware and software features of the three anti-hacking device platforms that we currently support for the CARAMEL project. Building on this preliminary information we will continue to work on features like secure boot and secure firmware update as well as secure Docker container technology and describe these efforts in the forthcoming deliverables D5.5 and D5.6 of WP5.

7 Annex: Secure Hardware Platform

This annex provides preliminary information that will be presented in greater detail in D5.5 “Secure Hardware Platform Specification” in order to help understanding the HSM integration steps described in this document. Due its preliminary nature some of the material presented here is still very at a high-level of abstraction and subject to change while the implementation work progresses.

7.1 Anti-hacking Device Overview

The CARAMEL anti-hacking solution is an important part of the project innovation. In this section, we have a deeper look on the general architecture and functionalities of it.

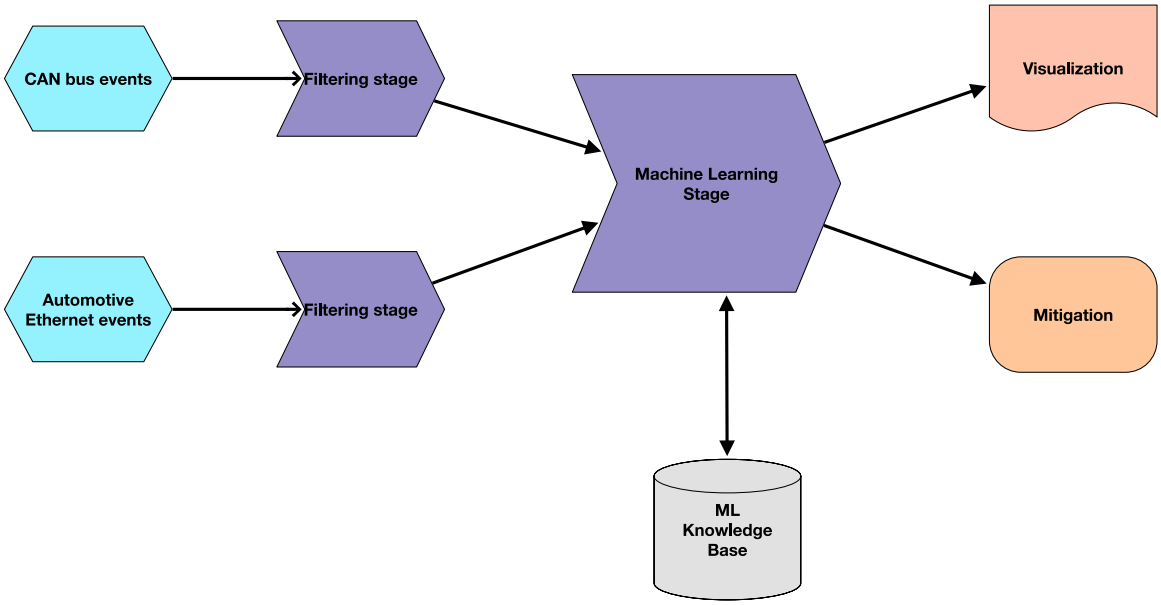

Figure 14: Machine Learning Pipeline

The anti-hacking device is a physical controller that is integrated into the car and acts as an attack detection device. In the Autonomous Mobility scenario its task is to run pre-trained ML models that work on the sensor data to detect anomalies that might point to malicious attacks. Additionally, the anti-hacking solution might be used for different functions in the context of the CARAMEL project, i.e. if needed it can ensure security for an embedded application platform. In this case, the software layer of the solution might be employed only. Further details about this approach will be presented in the rest of this document.

The anti-hacking device is connected to the busses in the car carrying the sensor data. It passively monitors the bus traffic (e.g. CAN bus frames) and extracts the raw sensor data.

Figure 14 shows the ML pipeline where raw data, e.g. from the CAN bus is pre-filtered and aggregated to make it suitable for the following machine learning stage to detect threats and attacks. Any security-relevant events are then forwarded to the visualization and mitigation components in the car.

The ML knowledge base (model) is pre-loaded into the anti-hacking device. The model will have been created offline on a more powerful system based on simulated and real-world training data.

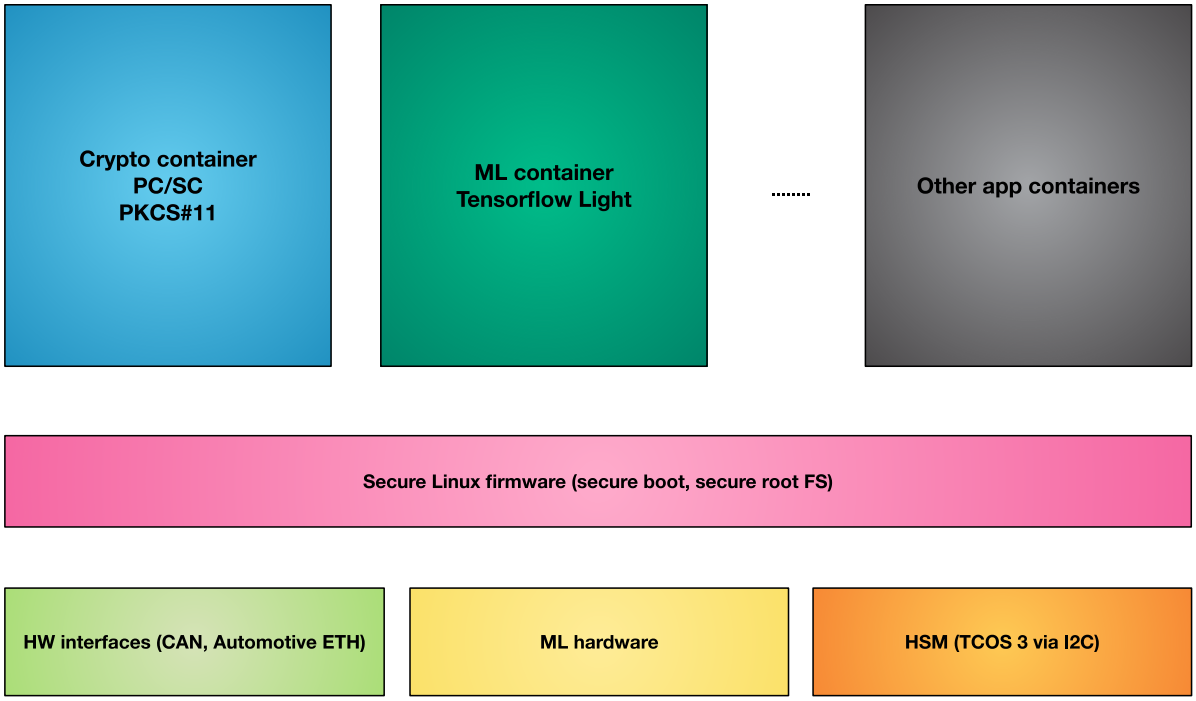

Figure 15: Anti-hacking Device Software Architecture

Figure 15 shows an overview of the software and hardware architecture of the anti-hacking device. From bottom-up the following components make up the anti-hacking devices:

- Hardware (HW) Interfaces: The anti-hacking device will be connected to the in-car systems via appropriate interfaces used in the automotive industry such as the CAN bus or Automotive Ethernet connections. For integration into development and simulation frameworks standard Ethernet will also be supported.
- The anti-hacking device will also support machine learning (ML) hardware. Since the anti-hacking device is based on the Coral Dev Board the Tensorflow Lite Processing Unit (TPU) is the hardware element to support ML. For a development and simulation configuration the Coral USB Accelerator will also be supported.
- HSM (hardware security module): To provide security-related functions of the anti-hacking device the hardware will integrate a Secure Element or HSM in the form of a TCOS (Telekom Card Operating System) embedded smartcard module that supports secure storage of private keys and different cryptographic operations.
- The anti-hacking device itself is based on an NXP Freescale i.MX8 processor that supports security functions such as hardware-assured boot.
- On this security hardware runs a Yocto-based firmware layer (a Linux embedded meta distribution).
- On top of this firmware substrate Docker-based application-specific containers can be loaded. Out-of-the box there will be crypto containers supporting the security functions of the anti-hacking device. ML workloads will be also be implemented as containers that have access to the underlying ML hardware as well as the crypto functions exported by the crypto container.
- The anti-hacking device could also act as a secure run-time environment for other functions as needed by the different use cases.

7.2 Coral Dev Board

7.2.1 Hardware Overview

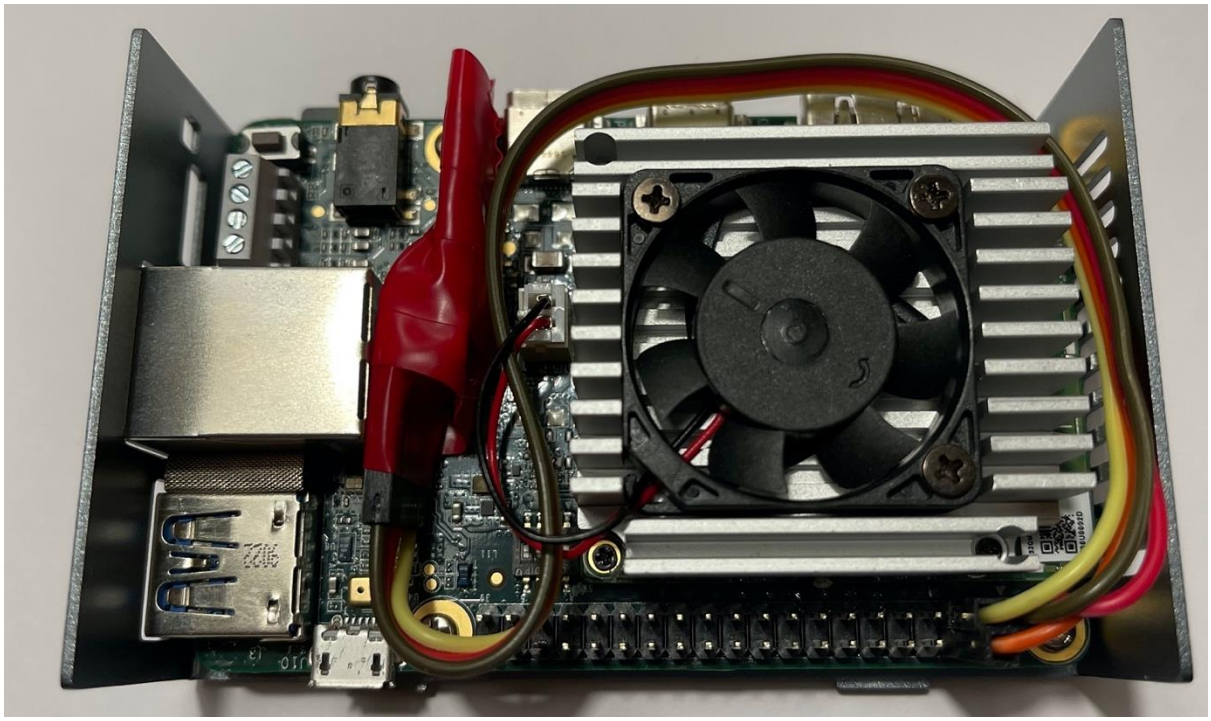

Figure 16: Anti-hacking Device Hardware with TCOS module via I2C

Figure 16 shows a picture of both the final target hardware - the Coral Dev Board[1].

The Coral Dev Board has the following hardware specifications:

- CPU: NXP i.MX 8M SOC (quad Cortex-A53, Cortex-M4F)
- GPU: Integrated GC7000 Lite Graphics.
- Coprocessor: Google Edge TPU.
- RAM: 1GB LPDDR4.
- Flash memory: 8GB eMMC.
- Connectivity: Wi-Fi 2×2 MIMO (802.11b/g/n/ac 2.4/5GHz) Bluetooth 4.1.
- Dimensions: 48 x 40 x 5mm.

The i.MX8 SOC includes advanced security features such as HAB (high-assurance boot) and CCAM (Cryptographic Accelerator and Assurance Module) that will support the security features of the Anti-hacking device. The firmware for the i.MX8 SOC will be created using the Yocto environment which is an industry-standard toolkit to create custom embedded firmware images in a reproducible manner. Our build process will support signed bootloaders and Linux kernel in order to prevent tampering with the anti-hacking device software and configuration.

The Coral Dev Board also has many connectivity options integrated on the board:

[1] https://coral.ai/docs/dev-board/get-started/

- Ethernet port (can be used for IP-based connections in a simulation and test environment, or to attach Automotive Ethernet adapters if needed)
- GPIO and I2C ports (used for connecting the HSM module, can be used for other purposes as well)
- USB port (used in the project to connect USB-to-CAN-bus converters)
- Wireless connectivity - Wi-Fi and Bluetooth

The Edge TPU processor integrated into the Coral Dev Board supports the execution of Tensorflow Lite models, performing 4 trillion operations (tera-operations) per second (TOPS), using 0.5 watts for each TOPS (2 TOPS per watt). The same Edge TPU is integrated into the USB Accelerator stick, so similar performance can be expected in the Anti-hacking Device simulation environment.

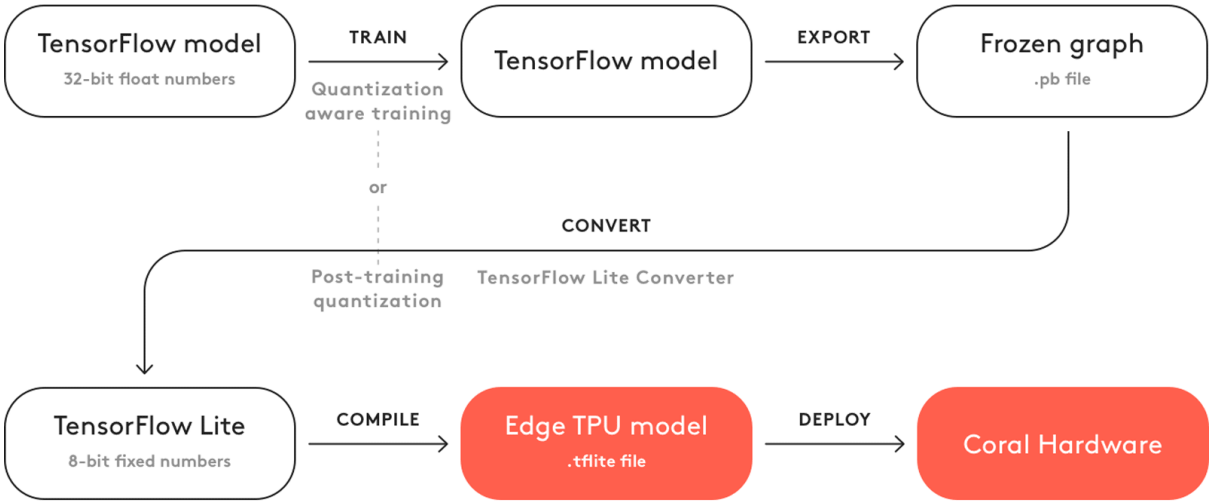

Figure 17: Conversion of Tensorflow model for use with Edge TPU

Figure 17 (source: compile-workflow.png) shows how TensorFlow models created by a machine learning process (e.g. running in the cloud or on project hardware) can be converted for use with either Coral Dev Board or the Coral USB Accelerator.

The I2C ports of the Coral Dev Board will be used to connect an HSM (hardware security module) based on the TCOS (Telekom Card Operating System) specification to act as an embedded Secure Element (eSE) and security anchor for the Anti-hacking device. The HSM is meant to support the following functions:

- Authentication of the Anti-hacking device for remote provisioning and updates
- Provide support for other CARAMEL use cases that need HSM functionality
- Authentication of the anti-hacking device against central systems such as Automotive SOC (Security Operations Centre) for event reporting and alerting

Figure 18: Coral Dev Board in Aluminum Case

Figure 18 shows the final integration of the Coral Dev with I2C module into an aluminum case. This configuration is suitable for deployment into vehicles and test environments.

7.2.2 Initial Software Installation

The firmware for the Coral Dev Board is specifically created for the CARAMEL project using a Yocto-based Linux firmware build process [11]. The details of the build process will be described in D5.5 and D5.6. The firmware is provided for installation on a 32 GB Micro SD card inserted in the Coral Dev Board.

First, install the official Mendel OS on the internal eMMC as described in [5].

The latest version of the Yocto-based firmware for the SD card built for the CARAMEL project is available via this link:

https://caramelx.mine.bz/caramelx-h2020-prv_dcs/coral/core-image-base-coral-dev.wic.gz

The following instructions assume that you use the “Dual-boot configuration for development purposes” described in section 7.2.3 and execute all the commands when booted into Mendel OS. If you use any other operating system (such as Linux or MacOS) you will have to adapt the commands appropriately.

First boot into Mendel OS and download the current Yocto-based firmware from the link above. Then execute the following commands to write the firmware to the Micro SD card (32 GB recommended):

```
zcat core-image-base-coral-dev.wic.gz | sudo dd of=/dev/mmcblk1 bs=4M
```

Then resize the root partition on the SD card to fill the rest of the available space:

resize2fs /dev/mmcblk1p2

This is needed to provide enough space for the Docker images on the card.

At the moment docker-compose is not properly set up in the provided firmware image. Please execute the following commands while connected to the Internet to install the missing dependencies:

```
wget "https://bootstrap.pypa.io/get-pip.py"
python3 get-pip.py
pip3 install "jsonschema<3,>=2.5.1"
cd /usr/lib/python3.7/site-packages/
rm -rf PyYAML-5.1.2-py3.7.egg-info
pip3 install "PyYAML<4,>=3.10"
pip3 install "requests!=2.11.0,!=2.12.2,!=2.18.0,<2.20,>=2.6.1"
```

This problem will be corrected in a forthcoming version of the Yocto firmware build.

7.2.3 Dual-boot Configuration for Development Purposes

For development purposes it is recommended to leave the boot switches as is into order to boot into Mendel OS first. To test the Yocto build you have to boot from the SD card, however. This can be achieved by interrupting the automatic uboot by pressing a key and issuing the following commands on the uboot command line (it make take some time until the kernel is loaded and booting, so don't power cycle but wait):

setenv bootdev 1

setenv bootcmd "ext2load mmc 1:1 ${loadaddr} boot.scr; source; boota mmc0 boot_a;"

saveenv

boot

Note that you must use the serial console (ie. the Micro USB port on the Coral Dev Board) connected to a PC to interrupt the boot process and enter the abovementioned commands.

Of couse, after the Yocto-based firmware has proven stable, it is possible to permanently switch to SD card boot by changing the DIP switch positions as follows:

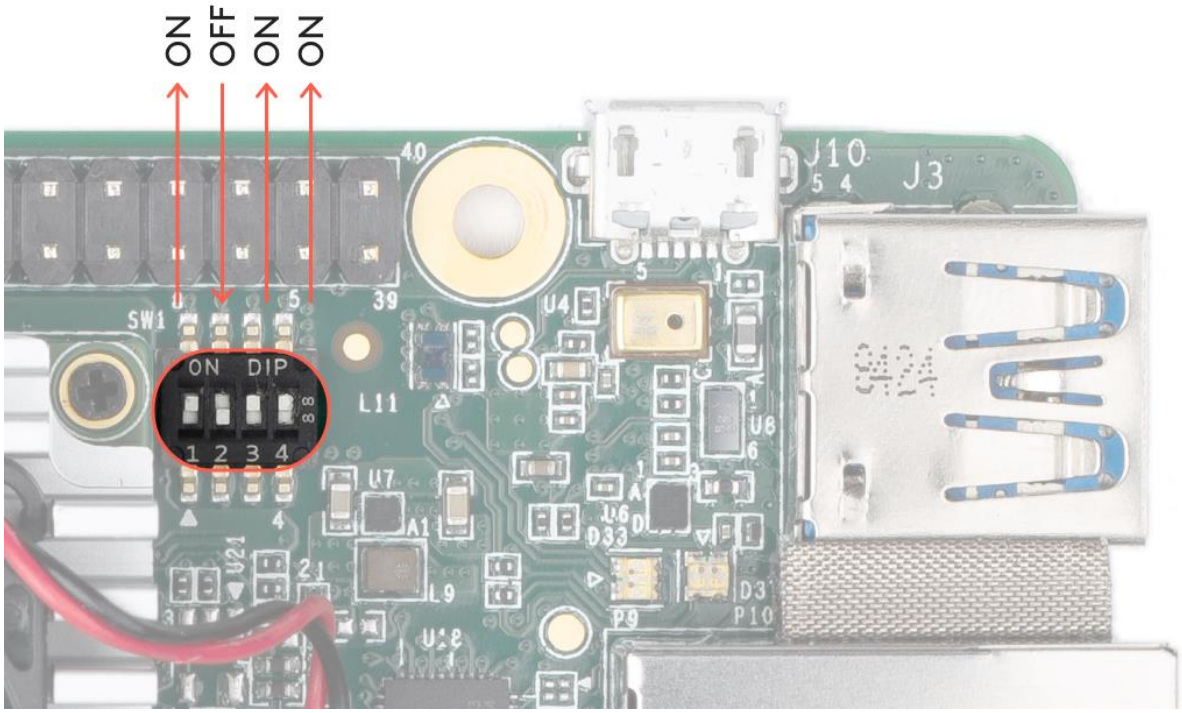

Figure 19: Coral Dev Board DIP switch positions for SD card boot

In order to boot from Mendel OS you then need to interrupt the uboot process again and issue these commands:

setenv bootdev 0

setenv bootcmd "ext2load mmc 0:1 ${loadaddr} boot.scr; source; boota mmc0 boot_a;"

saveenv

boot

7.3 NVidia Jetson AGX

7.3.1 Hardware Description

Figure 20: NVidia Jetson AGX Embedded Controller

Figure 20 shows the NVidia Jetson AGX embedded controller. The Jetson AGX is the most performant member of NVidia's Jetson range of devices. It features:

- GPU: 512-core Volta GPU with Tensor Cores
- CPU: 8-core ARM v8.2 64-bit CPU, 8MB L2 + 4MB L3
- Memory: 32GB 256-Bit LPDDR4x | 137GB/s
- Storage: 32GB eMMC 5.1
- DL Accelerator: (2x) NVDLA Engines
- Vision Accelerator: 7-way VLIW Vision Processor
- Encoder/Decoder: (2x) 4Kp60 | HEVC/(2x) 4Kp60 | 12-Bit Support
- Multiple USB connectors
- GPIO header with I2C

We have also integrated a 1 TB NVMe SSD to store large datasets locally on the device.

7.3.2 Software Installation

As of the time of this writing the NVidia provides a Jetson-specific firmware based on Ubuntu 18.04 that provides a uniform runtime and development platform for all Jetson devices.[6] The NVidia developer documentation [6] describes the installation processes of the firmware in detail.

Since the anti-hacking device functionality is based on Docker images and the Docker runtime is pre-installed in the Jetson firmware no further software installation is necessary in order to run the CARAMEL software stack on the device.

7.4 Kontron K-Box K-BOX A-330 MX6

7.4.1 Hardware Description

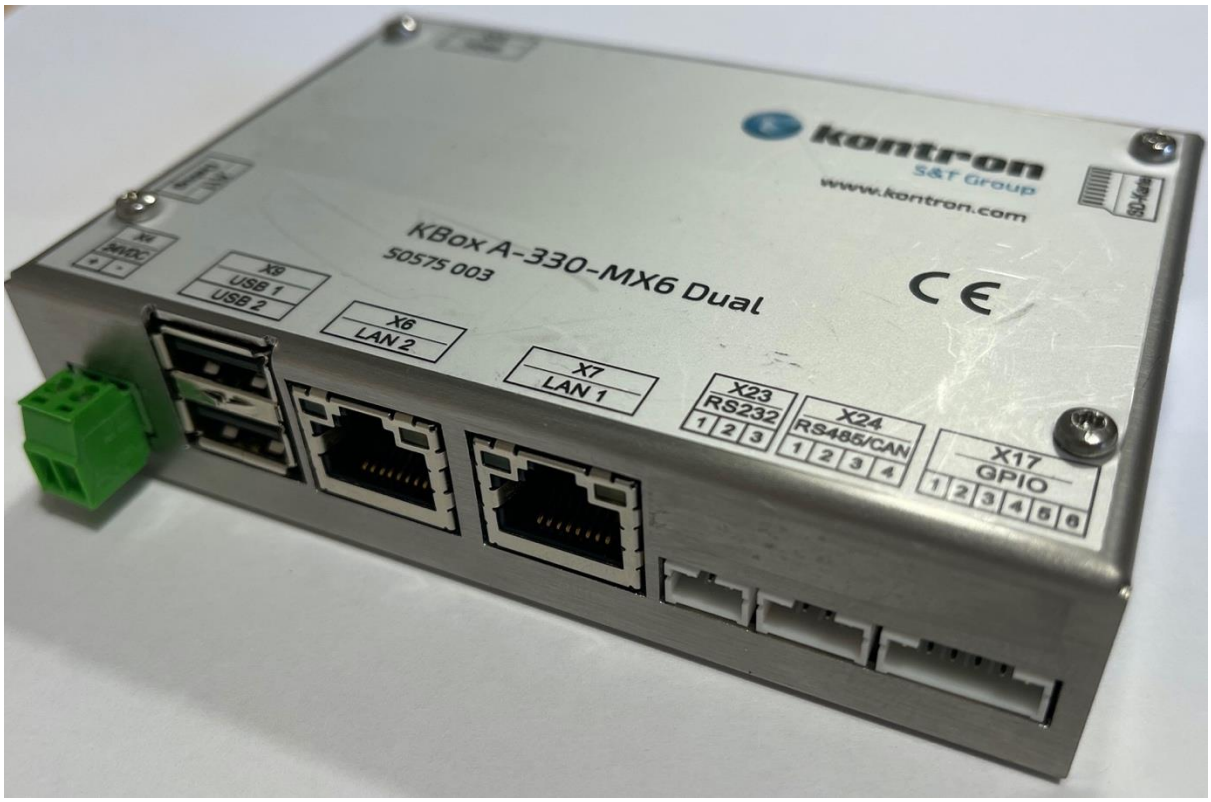

Figure 21: Kontron K-Box K-BOX A-330 MX6 Embedded Controller

CPU	NXP i.MX6 Dual Core/ULL Arm Processor
RAM technology Capacity	DDR3L 512 MByte (MX6-ULL)
Graphics format	1080p – 1920x1080

USB ports Ethernet ports Serial and other ports	2x USB 2.0 2x Fast Ethernet 1x RS232, 1x RS485 (switchable CAN) 4x Digital Out
Graphics port	HDMI
Internal storage External storage	4GB eMMC Micro SD slot
OS	Yocto
Construction Cooling Dimensions (HxWxD) Weight Mounting	Stainless steel Fanless 111 x 25 x76 mm Approx. 0,26 kg for mounting on 35mm mounting rail acc. to EN 60715
Operating temperature Storage temperature Relative humidity	0 °C to +55 °C (32 °F to 131 °F) -20 °C to +80 °C (-4 °F to 176 °F) 93% @ 40 °C, non-condensing
Power supply range	9..32 V DC

Table 3: Kontron K-Box K-BOX A-330 MX6 hardware specifications

Table 3 shows the hardware specifications of the K-Box A-330. The system has a dual-core ARM SoC with 512 MB of main memory (RAM) [3]. Since the 4GB internal eMMC might not be sufficient to host a large number of Docker images we have decided to use 32GB Micro SD cards to install the CARAMEL firmware load. This also allows us to install two images in parallel for the secure firmware update process we plan to implement for the anti-hacking device.

7.4.2 Software Installation

The default software installation is described in detail in [10].

The software build process is based on the Yocto firmware build system [11]. In order to implement pre-installed Docker containers and Docker container autostart as well as secure boot and secure firmware update, we have implemented additional recipes for the CARAMEL project that augment the Kontron base Yocto firmware build files. These will be described in a later deliverable in WP5.

The CARAMEL firmware image has to be installed to a Micro SD card to support the pre-installed Docker images (crypto and other containers) as well as the dual-boot setup for the secure and reliable firmware update process we plan to implement.

References

[1] European Commission – Grant Agreement Number 833611 - CARAMEL. 2019.

[2] CARAMEL – D5.1: Hardware Security Module Specifications. 2020.

[3] Kontron: KBox A-330-MX6 datasheet. Kontron, 2020.

[4] CARAMEL – D2.4: System Specification and Architecture. 2020.

[5] Coral: Update or flash the board. https://coral.ai/docs/dev-board/reflash/

[6] NVidia Jetson Developer Guide. https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/introduction.html#.

[7] Kontron: KBox A-330-MX6. https://www.kontron.com/products/systems/embedded-box-pc/kbox-a-series/kbox-a-330-mx6.html.

[8] Kontron: Description of the iMXceet Solo / Dual S Demoboard. https://docs.kontron-electronics.de/yocto-ktn/build-ktn-rocko/board-imxceet-solo-dual-s/.

[9] IETF: RFC 7030: Enrolment over Secure Transport. https://tools.ietf.org/html/rfc7030. 2013.

[10] Kontron Electronics: Quickstart - Kontron Electronics Docs - NXP i.MX6 (Rocko). https://docs.kontron-electronics.de/yocto-ktn/build-ktn-rocko/quickstart/.

[11] Yocto Project: The Yocto Project. https://www.yoctoproject.org.