TECHNISCHE UNIVERSITÄT WIEN

TTControl HYDAC INTERNATIONAL

Security Concept and Evaluation of an Off-Highway Electronic Control Unit

Diploma Thesis

Conducted in partial fulfillment of the requirements for the degree of a Diplom-Ingenieur (Dipl.-Ing.)

supervised by
Privatdoz. Dipl.-Ing. Dr.techn. Wilfried Steiner

submitted at the
TU Wien
Faculty of Electrical Engineering and Information Technology

by
Lukas Reier, BSc
Matriculation number: 01226448

Vienna, September 2020

Research Unit of Cyber-Physical Systems
A-1040 Vienna, Treitlstr. 3, Internet: cps.tuwien.ac.at

Abstract

Mobile machinery and vehicles used for agricultural, construction, mining, and material handling purposes are currently experiencing a rapid transition from exclusively mechanical to cyber-physical systems. In order to increase the efficiency and productivity of such off-highway vehicles, electronic components with advanced connectivity are being integrated. However, these electronic systems and their interfaces introduce new cybersecurity vulnerabilities. The off-highway industry is experienced in developing safe systems, but security is an emerging new field with less practice.
The aim of this master thesis is to closely examine an example use case for an off-highway electronic control unit (ECU). Based on the identified cybersecurity threats identified by a Threat Analysis and Risk Assessment (TARA), a security concept is developed, implemented and evaluated. The security requirements resulting from the risk assessment are refined and solutions to minimize the threats and increase cybersecurity are presented. Appropriate cryptographic primitives for such an embedded system are selected.
For the off-highway ECU an embedded software solution is developed. The ECU contains an automotive microprocessor with an additional security coprocessor. The software solution consists of two programs, one executed on the main core of the microprocessor and one running on the microprocessor's security coprocessor. The software implemented for this thesis aims to mitigate the security threats and to make use of hardware accelerations provided by the security coprocessor. The performance of the implementation is evaluated.

Kurzfassung

Mobile Maschinen und Fahrzeuge, die in der Landwirtschaft, im Baugewerbe, im Bergbau und bei der Materialhandhabung eingesetzt werden, erleben einen Übergang von ausschließlich mechanischen Systemen zu Cyber-Physical Systems (CPS). Die steigende Nachfrage nach Effizienz, Produktivität und Automatisierung für solche Off-Highway-Fahrzeuge erfordert die Integration elektrischer und elektronischer (E/E) Komponenten mit fortschrittlicher Konnektivität. Elektrische und elektronische Systeme, ihre Schnittstellen und ihre Kommunikation bringen jedoch neue Cybersecurity-Schwachstellen mit sich. Vergleichbar mit dem Automobilsektor hat die heutige Off-Highway-Industrie Erfahrung in der Entwicklung von sicheren Systemen (Safety). Cybersecurity ist jedoch ein neues Feld, wo weniger Praxis vorhanden ist. Das Ziel dieser Masterarbeit ist es, einen beispielhaften Anwendungsfall für ein elektronisches Steuergerät (ECU) für Off-Highway-Fahrzeuge zu untersuchen. Basierend auf den durch eine Bedrohungsanalyse und Risikobewertung (Threat Analysis and Risk Assessment, TA-RA) identifizierten Bedrohungen wird ein Sicherheitskonzept entwickelt, implementiert und evaluiert. Die aus der Risikoanalyse resultierenden Sicherheitsanforderungen werden konkretisiert und Lösungen werden vorgestellt. Geeignete kryptographische Algorithmen werden ausgewählt. Ein Demonstrator, der den Anwendungsfall darstellt, wird implementiert. Das Kernstück des Demonstrators ist ein Off-Highway-Steuergerät, das mit einem hochmodernen Automobil-Mikroprozessor ausgestattet ist. Der Mikroprozessor enthält einen zusätzlichen Sicherheitsprozessor mit Hardwareunterstützung für verschiedene kryptographische Funktionen.

Das praktische Ziel der Arbeit ist die Entwicklung einer Softwarelösung, die auf dem Steuergerät ausgeführt wird. Die Software zielt darauf ab, die Sicherheitsbedrohungen abzuschwächen und die Hardwarebeschleunigungen des Mikroprozessors zu nutzen. Die Implementierung wird hinsichtlich ihrer Ausführungszeit und Datendurchsatzes evaluiert.

Contents

List of Figures

List of Tables

Abbreviations

AAD	additional authenticated data.
AEAD	authenticated encryption with associated data.
AES	Advanced Encryption Standard.
ANSI	American National Standards Institute.
API	application programming interface.
AVB	Audio Video Bridging.
CAN	Controller Area Network.
CBC	cipher block chaining.
CCM	Counter with CBC-MAC.
CFB	cipher feedback.
CMAC	cipher-based message authentication code.
CPS	cyber-physical system.
CPU	central processing unit.
CTR	counter.
DES	Data Encryption Standard.
DoS	denial of service.
DRBG	deterministic random bit generator.
DRNG	deterministic random number generator.
DSS	digital signature standard.
E/E	electrical and electronic.
ECB	electronic codebook.
ECC	elliptic curve cryptography.
ECDH	elliptic-curve Diffie–Hellman.
ECDHE	ECDH ephemeral.
ECDSA	elliptic curve digital signature algorithm.
ECU	electronic control unit.
EMC	electromagnetic compatibility.
GCM	Galois/Counter Mode.
HMAC	keyed-hash message authentication code.
HMI	human-machine interface.
HSM	Hardware Security Module.
IV	initialization vector.
KMS	key management system.
MAC	media access control.
MAC	message authentication code.
MII	media-independent interface.
NPTRNG	non-physical true random number generator.
NVM	non-volatile memory.
OFB	output feedback.
PCB	printed circuit board.

PHY	physical layer.
PKC	public key cryptography.
PRNG	pseudorandom number generator.
PTRNG	physical true random number generator.
RAM	random access memory.
RNG	random number generator.
SECG	Standards for Efficient Cryptography Group.
SHA	secure hash algorithm.
SPI	Serial Peripheral Interface.
TARA	Threat Analysis and Risk Assessment.
TCP	Transmission Control Protocol.
TLS	transport layer security.
TRNG	true random number generator.
TSN	Time-Sensitive Networking.
UDS	Unified Diagnostic Services.
VLAN	virtual local area network.
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing.

1. Introduction

The off-highway sector includes mobile machinery and vehicles used for agricultural, construction, mining, and material handling purposes. The industry is experiencing an increasing demand for efficiency and productivity. Increasingly automated and connected machinery are a possible way to face the economical challenges. More efficient vehicles and processes enable a reduction of negative effects on the environment and on animal and human health. The emission of greenhouse gases and resource consumption can be lowered by providing sustainable technical solutions.

Connected and automated vehicles or machinery integrate computation with physical processes. Such systems are denoted cyber-physical systems (CPSs) [45, p. 1]. The behavior of CPS is defined by the cyber and physical components of the system. The physical process is controlled and monitored by embedded computers and communication networks. Terms such as IoT (Internet of Things), Industry 4.0, and Machine-to-Machine (M2M) are related to the expression CPS. [45]

The ECSEL AFarCloud (Aggregate FARming in the CLOUD) is a European project providing a distributed platform for autonomous farming [52]. Agricultural cyber-physical systems such as UAVs (Unmanned Aerial Vehicles) and autonomous ground vehicles are integrated within the system. The communication is in real-time. To measure the conditions of livestock and plants, intelligent sensors are used. The gathered data is uploaded to the cloud and further analyzed with big data and real-time data mining techniques. Farm management software makes the data accessible and enables the farmers to set the mission goals for the autonomous machinery. Novel precision farming and automation techniques are developed in order to increase the efficiency, the productivity, as well as to reduce farm labor costs. Precision farming is a paradigm where pesticides and fertilizers are applied precisely only where needed instead of over a large area.

The transition of off-highway vehicles from exclusively mechanical systems to CPSs brings new challenges. Secure and private communication between the machines, the cloud, and the operators is required. Cybersecurity threats jeopardize the safe and reliable operation of the CPS. Therefore, a sector experienced with building safe systems is challenged to focus also on security aspects.
For cars, which have similar E/E architectures, it has already been shown that advanced attacks are feasible. In 2015 a team of researches has demonstrated on a Jeep Cherokee that

they were able to control physical actions of the car by gaining access to its remote connection [36]. The researchers managed to send CAN messages via remote exploitations. Breaking the cyber components led to control over the physical components of this CPS. Steering and breaking functionality of the vehicle could be affected over the cellular network. The found issues are not limited to one model but affect many vehicles produced by Fiat Chrysler Automotive. 1.4 million cars have been recalled as result of the findings.
In 2017 another team of researchers investigated the security of a Volkswagen Golf GTE and an Audi A3 e-tron [43]. The goal was to analyze if the driving behavior of the cars could be influenced via their internet connection. The in-vehicle infotainment system could be remotely compromised and arbitrary CAN messages could be sent. In this preliminary research access to the vehicle's central screen, speakers and microphone was achieved. A gateway between the compromised CAN bus and the critical CAN bus prevented the team from affecting the driving behavior.
In 2018 another research team found vulnerabilities in several BMW models [49]. Via the USB, OBD and cellular network interface the vulnerabilities could be exploited. Not only local but also remote access to the vehicle electronics was gained. Unauthorized diagnostic requests could be executed by sending malicious CAN messages. The demonstrated potential and feasibility of such attacks makes mitigation necessary.

The thesis examines how an off-highway use case can be analysed in the context of cybersecurity. Based on the identified threats, solutions are elaborated. Furthermore, the question on how to efficiently implement the provided solutions on an embedded system is addressed. The embedded system in this thesis in an electronic control unit (ECU) for off-highway vehicles. Additionally, the questions on how to conduct security tests as well as on how to verify cryptographic algorithms are answered.

1.1 Structure of the Thesis

This thesis is structured as follows. Chapter 2 contains background information. Security standards which are used in the automotive sector and also applicable to the off-highway sector, are presented. The security standards section is followed by an introduction to modern cryptography. The architectural specification of a hardware-based security concept for automotive communications is described. The security concept is called Hardware Security Module (HSM). After the introduction of the HSM, suggestions on how to verify the correctness of cryptographic algorithms are provided. Security testing is the final topic described in chapter 2. Chapter 2 will be concluded by the final topic of security testing which includes a listing of available tools.
Chapter 3 treats the case study of an example use case for an off-highway electronic control unit (ECU). The cybersecurity threats to the example use case are analysed and solutions to reduce the risk are presented. The onboard architecture is reworked and a software solution for the ECU is presented. Secure diagnostic communications between the vehicle and an external device are enabled by the software solution.
Chapter 4 describes the technical specification of the off-highway ECU from the example use

case. The focus lies on the microprocessor's HSM.

Chapter 5 describes the embedded software implemented in the course of the thesis. The software is developed for the previously described off-highway ECU. The program executed on the secure core and the interface functions available to the main core of the microprocessor are specifically developed for this thesis. Additionally on the main core a demonstration application integrating the HSM interface functions is developed. Single components of the developed embedded software are described in this chapter.

The performance evaluation of the implemented software is presented in chapter 6. A comparison with a software-only implementation is conducted.

Chapter 7 concludes the thesis and provides an outlook for further improvements of the presented solution.

2. State of the Art

At first, this chapter introduces several security standards for the automotive domain (section 2.1). In the second section an introduction to the later implemented cryptographic principles (section 2.2) is provided, followed by the architectural specification of a security module integrated in automotive microprocessors (section 2.3). The security module is called Hardware Security Module (HSM) and provides secure and efficient cryptographic algorithms. The introduction of the HSM is followed by two suggestions on how to verify the correctness of cryptographic algorithms (section 2.4). The final section of the chapter introduces techniques for security testing with a special focus on penetration tests (section 2.5).

2.1 Security Standards for Automotive

In the sector of information technology (IT) several established security standards exist. The international standard Common Criteria for Information Technology Security Evaluation (ISO/IEC 15408), referred to as CC or Common Criteria, defines a common set of requirements to allow comparability between the results of independent security evaluations [42]. The standard is focused on the security functionality of IT products. The hardware, firmware and software of IT products are in the scope. The CC offers guidelines for development, evaluation, and procurement of such IT products with security functionality.
The ISO/IEC 27000-series is another set of standards focusing on best practice recommendations on information security management [31]. The focus lies on IT systems such as databases, corporate networks, and servers. For the industrial sector the standard IEC 62443 (Network and system security for industrial-process measurement and control) is available [31].
Such standards, although they are not intended for the automotive sector, are relevant for production and back-end IT systems [55].

For the automotive and off-highway sector the center of attention lies on security in combination with safety. For automotive cybersecurity engineering the SAE J3061 guidebook was published. This recommended practice defines a lifecycle process analogous to the one defined in ISO26262 [55]. ISO/SAE 21434 (Road vehicles - Cybersecurity engineering) cancels and supersedes J3061 [60]. The standard is under development with the current status of a Draft International Standard (DIS). ISO/SAE 21434 aims to support organizations to define cybersecurity policies and processes, manage cybersecurity risks, and foster a cybersecurity culture [60].

Other activities regarding the standardization of automotive security aspects are conducted by ITU-T SG17 Q13 (Security Aspects for Intelligent Transport System) and United Nations Economic Commission for Europe (UNECE) [55]. ITU is focused i.a. on V2X (vehicle to X) communication, in-vehicle system security, automotive Ethernet security, and secure updates. UNECE works on future regulations regarding vehicular cybersecurity and over-the-air updates in context of type approval. Drafts are already available (e.g. UNECE WP29 [67]). ISO is additionally working on extended vehicle web services security (ISO 20078-3 and ISO TR 23791) and security certificate management (ISO 20828) [55]. SAE is also working on requirements for vehicular hardware-based security and on guidance regarding OBD (On-Board Diagnostics) port security [55].

2.2 Cryptography

This section discusses digital cryptographic solutions which are later implemented in soft- and hardware. Cryptography is a fundamental cybersecurity tool to provide, amongst other properties, confidentiality and integrity [27, ch. 1.3.1].

2.2.1 Symmetric Key Cryptography

Symmetric key cryptography uses the equivalent key for encryption and decryption. To establish an encrypted communication channel both parties have to possess the same secret key (symmetric key). The unencrypted data is denoted as plaintext and the encrypted result is denoted as ciphertext. By using the decryption function, the plaintext can be obtained from the ciphertext.

Two different types of ciphers can be differentiated: stream and block ciphers. Stream ciphers and block ciphers are different symmetric key ciphers. Stream ciphers produce a continuous output, the key stream, which is combined with the plaintext to generate the ciphertext. Examples for stream ciphers are RC4, A5/1, and Salsa20. The German Federal Office for Information Security (BSI) recommends no stream cipher [63, ch. 2.2].
Block ciphers encrypt plaintext blocks with a fixed length. The resulting ciphertext is also a block with fixed length. Block ciphers are the digital successors of the codebook. The key configures the codebook and therefore determines the output. An example for a modern block cipher is the Advanced Encryption Standard (AES). AES is the successor of the Data Encryption Standard (DES). The weakness of DES is a small key size (56 bit) that makes brute force attacks feasible. AES is standardized by the United States National Institute of Standards and Technology (NIST) [3]. The block length of AES is 128 bits. Cryptographic key lengths of 128, 192, and 256 bits are standardized. According to the standard, at least one of the key lengths shall be supported by an implementation of the algorithm. The length of the key sequence for algorithm implementations is noted in the name, e.g. AES-128 stands for 128 bit key length. The AES algorithm is recommended by the German BSI [63, ch. 2.1].

Block Cipher Modes of Operation

If the length of the plaintext is greater than the block length of a block cipher, the encryption requires splitting up the input. The resulting input chunks are eventually padded and fed into a block cipher. A mode of operation "is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream" [48, p. 213]. NIST specifies different modes of operation for use with any (recommended) block cipher [4].

The Electronic codebook (ECB) mode is a basic mode of operation which splits the plaintext into equally sized blocks (padded if needed) and encrypts them with the same key. The major weakness of ECB is that an identical plaintext block always yields an identical ciphertext block.
A more advanced mode of operation is cipher block chaining (CBC). Each resulting ciphertext block is XORed (exclusive-or operation) with the next plaintext block. The XORed result is then encrypted to generate the next ciphertext block. The block ciphers are "chained together". CBC also provides an additional input, the initialization vector or initial value (IV). The first plaintext block is XORed with the IV. The IV should be randomly selected but is not required to remain secret. Every time a new IV is used the ciphertext changes even if plaintext and key remain the same.
The counter (CTR) mode generates a keystream which is then XORed with the plaintext. The generated keystream is used in the same way as the keystream from a stream cipher. The keystream is not dependent on the plaintext and can be precomputed. A nonce (number used once) together with a counter value form the IV. The nonce remains constant while the counter value is incremented for each block. Encryption and decryption are the same operation and parallelizable.
The CTR and CBC modes are recommended by the German BSI [63, ch. 2.1.1]. Additional examples for modes of operation are cipher feedback (CFB) mode and output feedback (OFB) mode.

Authenticated Encryption

Protocols such as TLS use encryption and authentication simultaneously. The German BSI states that "no decryption or other processing should be performed for unauthenticated encrypted data" [63, p. 22-23]. Authenticated encryption describes encryption systems protecting confidentiality and authenticity (integrity) of data [48, p. 402]. NIST specifies two modes of operation for authenticated encryption, the Counter with CBC-MAC (CCM) mode [6] and the Galois/Counter Mode (GCM) [9].

The GCM mode consists of an adapted CTR mode (GCTR function) and the GHASH function. The GHASH function is a keyed hash function. GCM is designed to enable high-throughput implementations in software and hardware [9, p. 2]. GCM provides authenticated encryption with associated data (AEAD). As input for the encryption plaintext, additional authenticated data (AAD), and an IV are taken. The output of the algorithm is the ciphertext and the authentication tag. The ciphertext is the encrypted plaintext. Not encrypted, but integrity-protected by the authentication tag, is the AAD. The tag protects the ciphertext and AAD in a sense that both accidental and intentional, unauthorized modifications are detected. The confidentiality of the plaintext is protected by encryption using a block cipher (e.g. AES). GCM specifies an authenticated decryption function which does not only decrypt the ciphertext but also verifies the authentication tag. The GCM as well as CCM mode are recommended by the German BSI [63, ch. 2.1.1].

2.2.2 Asymmetric Key Cryptography

Asymmetric key cryptography, also known as public key cryptography, uses different keys for encryption and decryption. The key pair consists of a public and a secret/private key. Each node publishes its public key which then can be used to encrypt data. Using the private key nodes can decrypt received data. This is the basic principle of asymmetric encryption. Public key cryptosystems are based on trap door one-way functions [48, p. 90]. Apart from encryption, the establishment of shared secrets as well as the generation and verification of digital signatures are additional use cases for asymmetric cryptography. A digital signature is generated using the private key and can be verified using the public key. More details on different public-key cryptosystems are given below.

Diffie-Hellman Key Exchange

The Diffie-Hellman (DH) algorithm allows the establishment of a joint secret key. Two communicating entities agree on a shared secret. DH is based on the discrete logarithm problem.
An enhanced variant of the static DH algorithm is the ephemeral DH algorithm. Ephemeral DH provides (perfect) forward secrecy. Meaning that the decryption of earlier messages is not possible if an adversary gets one key. To do so the ephemeral DH uses session keys which are periodically renewed. The session keys are denoted as ephemeral (temporary, one-time) secret keys. The secret intermediate values to derive the session keys are deleted right afterwards.

RSA

RSA (Rivest Shamir Adleman) uses the factoring of large integers as one-way function. Applications of the RSA scheme are digital signatures and encryption. Due to the bad performance in signature generation, the long-time security aspect, and the great length of signatures RSA is not considered suitable for automotive use cases [26].

Elliptic Curve Cryptography (ECC)

Applications for Elliptic Curve Cryptography are key exchange and digital signatures. ECC makes use of elliptic curve arithmetic. An elliptic curve is the graph of a function in the

form of $y^2 = x^3 + ax + b$ in combination with a special point at infinity [27, p. 103]. Point addition operations on such a curve are defined by the EC arithmetic and are used to establish a private and public key pair. Different representation variants of elliptic curves are the Weierstrass-Notation, the Montgomery-Notation and the (twisted) Edwards-Notation. For each notation standardized curve parameters are available. An advantage of ECC is that it offers, compared to RSA, an equal level of security with far smaller key sizes [27], [48], [56]. ECC is suitable for resource constrained devices [27, p. 103] such as embedded systems. According to the EVITA project, "ECC clearly outperforms every public-key cryptosystem that uses modular exponentiation for the same effort in hardware" [22, p. 237].
Examples for EC-based digital signature schemes are the elliptic curve digital signature algorithm (ECDSA) and the Edwards-curve Digital Signature Algorithm (EdDSA). The static and ephemeral form of Diffie-Hellman Key Exchange (ECDH and ECDHE) are EC-based key exchange protocols.

Protocols such as TLS use asymmetric encryption to authenticate the communication partner and to establish a shared secret key. To encrypt and decrypt the bulk data, faster symmetric encryption algorithms are used together with the established secret key.

2.2.3 Message Authentication Codes

Message authentication codes (MACs) can be used to ensure the integrity of data. The technique generates a small fixed-size block of data using a secret key [48, p. 388]. This cryptographic checksum, or MAC, is appended to a message. The receiver verifies the MAC of the associated message using the secret key. The application of a MAC is not limited to messages but can be used for integrity protection of any kind of digital data. Two common techniques to generate MACs are described below.

Keyed-Hash Message Authentication Code (HMAC)

HMAC algorithms are based on cryptographic hash functions (see section 2.2.4). A secret key is added to the generation of the cryptographic checksum. Otherwise anybody could forge the MAC. NIST [11, p. 4] specifies the HMAC generation as the following operation:

$$MAC(text) = HMAC(K, text) = H(\ (K0 \oplus opad)\ ||\ H((K0 \oplus ipad)||text)\ )$$

where K is the key, $K0$ is the pre-processed key, $opad$ and $ipad$ are the outer and inner padding, $text$ is the data on which the HMAC is calculated, and H is the selected hash function. The exclusive-or operation is noted as $\oplus$ and the concatenation as $||$. For H every NIST-approved hash function can be used (e.g. SHA-256).

Cipher-Based Message Authentication Code (CMAC)

CMAC algorithms are based on symmetric key block ciphers such as AES. The same operation is used for encryption and authentication. An example for an older, now obsolete, CMAC algorithm is the Data Authentication Algorithm (DAA) based on DES [48, p. 399]. The block cipher is operated in CBC mode and the last "ciphertext" output is used as data authentication code. An improved version of the DAA is standardized by NIST as CMAC algorithm [38].

According to NIST [38, p. 1] and IETF [8, p. 1] both the CMAC and HMAC achieve similar security goals leaving the decision upon the availability of either hash functions or block ciphers on the respective system. The German BSI considers the HMAC and CMAC as secure as long as secure hash functions, secure block ciphers and a key above a minimum length are used [63, ch. 5.3].

2.2.4 Cryptographic Hash Functions

Cryptographic hash functions take an input of arbitrary length and generate an output of a fixed length. These functions have to fulfill certain requirements. If, for example, the input changes for one bit at least half of the bits of the output have to change. Additionally it must be computationally infeasible to find two inputs which generate the same output. Nevertheless, there must be collisions because the output space is smaller than the input space. [48, p. 41]
In contrast to a cryptographic hash function, it is easy to construct collisions for an arbitrary given CRC (cyclic redundancy check) [48, p. 132]. CRCs and similar techniques are designed to detect accidental errors but intentional, unauthorized modifications can remain undiscovered.
Hash functions are used e.g. for the HMAC, in pseudorandom number generators, and for ECDSA signatures. For ECDSA the message is hashed first and then the signature is generated.
NIST specifies and recommends secure hash algorithms (SHAs) in FIPS 180-4 [34] and FIPS 202 [35]. The SHA-3 hash functions from FIPS 202 are intended to supplement the SHA-1 and SHA-2 hash functions from FIPS 180-4. According to the German BSI the SHA-256, SHA-512/256, SHA-384, SHA-512, SHA3-256, SHA3-384 and SHA3-512 mechanisms are considered to be cryptographically strong [63, ch. 4.]. SHA-1 and SHA-224 are not recommended [63, ch. 4.].

2.2.5 Random Number Generators

In cryptography random numbers are used as symmetric keys, for Diffie-Hellman key exchange, as RSA key pairs, and for ECDSA signatures. Cryptographic random numbers do not only have to be statistically random but must also be unpredictable [27, p. 146]. The Shannon Entropy is a characteristic value for unpredictability.

Random number generators (RNGs) usually consist of a non-deterministic part and a deterministic part [24, ch. 2.1.2]. The non-deterministic part is an entropy source that generates non-predictable data. The deterministic part generates the output by post-processing the generated non-predictable data. It is also possible to omit either the deterministic or the non-deterministic part. Categories of RNGs are physical true random number generator (PTRNG), non-physical true random number generator (NPTRNG), deterministic random number generator (DRNG), and Hybrid RNG. [24]

The entropy source (non-deterministic) is the core of a PTRNG which is used to generate raw random data. The generator may include additional post-processing (e.g. to increase entropy or correct a bias) of the raw data stream. The entropy source is based on a physical microscopic random process. Examples for such processes are radioactive atomic disintegration, shot entropy of a diode, free running oscillators, and the thermal resistive entropy. [24]
The entropy sources of NPTRNGs are external signals. The concept of randomness is based on the lack of information about processes and their outcomes. Examples for such processes are disk I/O operations, interrupts, thread IDs, current local time, and human interaction such as mouse movement and key strokes. The best known implementation of a NPTRNG is the `/dev/random` in Linux operating systems [63, p. 63].
DRNGs use deterministic algorithms to generate a random number from a random seed. Optional additional external input values can be used to improve the output quality. Specifications for DRNGs based on hash functions and block ciphers are available [33].
Hybrid RNGs consist of a deterministic and a non-deterministic part. An entropy source is used to seed the deterministic RNG.

2.3 EVITA Full HSM

This section provides an architectural description of the Hardware Security Module (HSM) specified by the EVITA (E-Safety Vehicle Intrusion Protected Applications) project. An HSM device enables a secure and efficient implementation of cryptographic algorithms.

The EVITA project [23] was co-funded by the European Commission and coordinated by the Fraunhofer Institute for Secure Information Technology. Numerous companies from the automotive industry were partners of the project. The project lasted for 42 months and was completed in December 2011. The objectives were ”to design, to verify, and to prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering and sensitive data are protected against compromise” [23].
The architectural specification of a Hardware Security Module (HSM) resulted from the EVITA project and is documented in Deliverable D3.2: Secure On-board Architecture Specification [22]. The specified architecture aims to provide a hardware-based security mechanism to protect secrets (e.g. cryptographic keys) and to securely execute computations (e.g. encryption algorithms). Alternative cryptographic hardware concepts are, for example, SHE (Secure Hardware Extension), TPM (Trusted Platform Module), and smartcards [22].
Fig. 2.1 illustrates the general architectural topology of the HSM. The left (red) block is the HSM coprocessor which communicates with the application core via interrupts and a shared RAM area. The application CPU (yellow block on the right side) has the ability to trigger interrupts on the HSM and vice versa. Data is exchanged using a shared area in the application core's RAM. The application core has PFlash and DFlash non-volatile memory

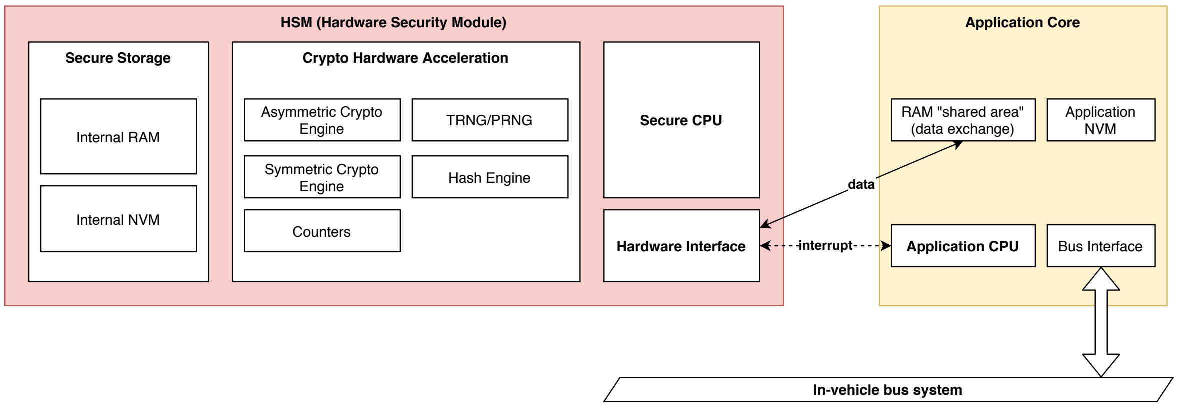

Figure 2.1: HSM architecture (adapted from [22, p. 31])

(NVM) to store the application and application-relevant data. Vehicle bus interfaces are also accessed by the application core. The HSM has a hardware interface module to communicate with the application core. A secure CPU is the centerpiece of the HSM. A dedicated secure storage in form of volatile (RAM) and non-volatile memory (PFlash/PFlash) is also provided. The HSM module executes all cryptographic applications which include symmetric and asymmetric encryption/decryption, symmetric integrity checking, digital signature verification/generation and random number generation [22]. The HSM has cryptographic hardware acceleration for asymmetric/symmetric cryptography, random number generation, hash generation, and counters. The random number generator is either a pseudorandom number generator (PRNG) seeded with a true random number generator (TRNG) or an externally seeded PRNG. The counters are used as secure clocks.
According to specification, it is necessary to place the HSM on the same chip as the application core. The integration on the same chip mitigates the problem of eavesdropping on wires which run along the ECU's PCB [22, p. 31]. Furthermore, the focus of the specification lies on flexible solutions because automotive microcontrollers are used for ”more than 20 years” [22, p. 31]. Hardware-only solutions do not offer enough flexibility to adapt to possible new requirements, consequently the secure CPU must be programmable.
To address different cost segments and use cases, the EVITA HSMs are specified in three variants: full, medium, and light. Full EVITA HSMs are intended for V2X (vehicle-to-everything) communication, medium EVITA HSMs for on-board inter-ECU communication, and light EVITA HSMs for on-board communication with sensors and actuators [22]. Medium and light EVITA HSMs contain a subset of the features of the full EVITA HSM. Full EVITA HSMs support fast asymmetric and symmetric cryptographic operations, key storage, and hash generation.
The following building blocks are provided by an EVITA full HSM [22, p. 33-34]:

- **ECC-256-GF(p):** high-performance 256-bit elliptic curve cryptographic engine using NIST approved curve parameters [28]
- **Whirlpool:** hash function based on Advanced Encryption Standard (AES)
- **AES-128:** symmetric block encryption with 128-bit key in different modes of operation (i.e. ECB, CBC, GCM or CCM) [3]

- **AES-PRNG:** pseudorandom number generator seeded by an internal true random number generator
- **Counter:** simple secure clock alternative. At least 16 monotonically increasing 64-bit counters.
- **CPU:** internal CPU handling non-time-critical cryptographic functionality. The proposed CPU is a ARM Cortex M3 or similar.
- **RAM:** volatile memory for e.g. intermediate values and for variables. At least 64kB.
- **NVM:** non-volatile memory for e.g. internal keys, certificates, counter values. At least 32kB (+10kB ROM)
- **HW-API:** secure hardware interface between the application CPU and the HSM's functionality

It is stated that "the final set of cryptographic building blocks will be detailed in the implementation phase" [22, p. 34].

2.4 Verification of Cryptographic Algorithms

This section suggests methodologies for verification of implemented cryptographic algorithms. Verifying the correctness of the result of a cryptographic algorithm is challenging due to the algorithm's nature. Ciphertexts, message authentication codes, signatures, hash digests, and especially random numbers allow no trivial assessment whether they are correct or not. For other functions, in contrast, it might be possible to check the result for plausibility, allowing a fast initial assessment of the correctness. E.g. a calculated target position for a hydraulic valve can be assessed by having knowledge about the use case. A hash algorithm in contrast generates an apparently unrelated output where small changes in the input produce a completely different digest. High quality encryption algorithms are also designed to generate seemingly random outputs avoiding any disclosure of the plaintext. The properties of such algorithms require advanced verification techniques. During the development and testing phase of the HSM application (described in section 5.5), the implementations of the single cryptographic primitives are verified by using the following techniques:

- Comparing results and intermediate values with test vectors
- Comparing results with scripting language implementations

In the following section the two techniques are outlined in more detail.

2.4.1 Test Vectors

Institutions such as IETF or NIST provide test vectors for the standardized algorithms. Such test vectors list intermediate and/or final results for a given set of input values.
For the AES-CMAC examples are listed in the RFC 4493 [8, ch. 4]. The CMAC algorithms

specifies the generation of subkeys. For a given example key the resulting subkeys are provided. Such test vectors for intermediate results allow the step-by-step verification of the implemented algorithm. Subsequently, example messages with different lengths and the resulting MACs are listed. Certain algorithms such as AES-CMAC can handle input messages with arbitrary lengths but split the messages into blocks with a fixed length. Incomplete block segments are padded with a defined pattern. It is important to perform tests with different message lengths, in order to detect errors in the padding code.
For AES-based block cipher encryption modes such as ECB, CBC, CTR, and GCM test vectors are available at the website of NIST [37]. The website also provides test vectors for NIST-standardized digital signatures (e.g. ECDSA with SHA-2), hash algorithms (e.g. SHA-256), random number generators (e.g. CTR_DRBG), and message authentication codes. The Standards for Efficient Cryptography Group (SECG) published test vectors for ECC algorithms, such as ECDSA and ECDH [2].

2.4.2 Scripting Languages

The described test vectors allow to verify the correctness of results for a subset of input values. Scripting languages are suitable for fast software prototyping. Concurrent to the development of the embedded software, test programs are coded in a scripting language. To verify the implementations described in section 5.5, Python (version 3.7) with the `cryptography` (version 2.8) package [74] is used. The written Python code can be used as a starting point for further test automation. Python-based test systems, such as described in [50], can be extended with test cases from the verification process.

The `cryptography` package offers high level recipes and low level interfaces to cryptographic primitives. The high level recipes include Fernet and X.509 public key infrastructure. Fernet is an implementation of a symmetric authenticated encryption algorithm based on AES-128 in CBC mode and SHA-256. X.509 is an ITU-T standard for digital certificates in a public-key infrastructure. The standard is also published as ISO/IEC 9594-8 [44]. X.509 certificates are widely used in the TLS protocol to authenticate servers. `Cryptography` provides functions to generate self-signed certificates, certificate signing requests etc.
The relevant layer to verify the implemented algorithms is the lower level interface to cryptographic primitives. The package supports symmetric encryption (AES in different modes such as ECB, CBC, CTR, GCM etc.), asymmetric algorithms (digital signatures, key exchange), message authentication codes (CMAC, HMAC, Poly1305), and message digests (SHA-2, SHA-3, etc.). The asymmetric algorithms module supports the generation and verification of digital signatures with ECDSA, Ed448, and Ed25519. RSA is also supported. Key exchange algorithms such as Diffie-Hellman key exchange, ECDH, X448, and X25519 are supported. Elliptic curve parameters for NIST prime curves (e.g. secp256r1), NIST binary curves (e.g. sect571k1), and Brainpool curves (e.g. brainpoolP384r1) are included.
The set of supported algorithms allows to verify the results of the implementations for arbitrary input values. A drawback of the ECDSA and ECDH implementations, when it comes to testing, is that the functions are internally seeded with a random number. For every

generation the result is different and the function interface does not provide a parameter to manually set the random number. For real use cases it is beneficial to prevent the bypass of the random seed because the long term private key can be obtained from several signatures with identical random numbers. But for test cases, setting the random seed to a distinct value is required to guarantee comparable results.

2.5 Security Testing

According to ISO/SAE 21434 [60], component testing should be performed to discover unidentified vulnerabilities. Penetration testing, vulnerability scanning and fuzz testing are listed as examples for test methods. Fuzz testing applies large amounts of random data to the system input to find vulnerabilities and weaknesses [60]. The test is usually performed in a semi-automated or automated way. Overflows, segmentation errors, and heap errors potentially causing cybersecurity issues can be discovered with Fuzz testing.
Vulnerability scanning assesses and quantifies the exploitation risk of vulnerabilities [60]. Checklists of known vulnerabilities are used to perform passive and active scans. An example for a passive scan is to look for exploitable coding errors. Active scanning can be performed for access to memory, file systems, network protocols, or host processes.
Fuzz testing and vulnerability scanning can be used as techniques for penetration testing. In the following section penetration testing is described and tools are presented.

2.5.1 Penetration Testing

Penetration testing, often abbreviated pen testing, is defined by NIST as ”security testing in which accessors mimic real-world attacks to identify methods for circumventing the security features of an application, system, or network” [13, ch. 5.2]. Penetration testers look for vulnerabilities on real systems and often use tools and techniques commonly used by attackers [13]. The goal of such tests can be to assess the system’s tolerance against real-world attack patterns, the sophistication required to compromise the system, which additional countermeasures could mitigate the threats, and/or what is the defenders ability to detect attacks and respond properly [13]. According to NIST, penetration testing could also include non-technical attack methods. Such methods (e.g. social engineering) are not treated in this thesis. Penetration testing aims to find vulnerabilities that can be exploited in order to gain privileged access, take over control, expose privileged data, or cause a malfunction [60]. White, gray and black box tests can be performed, depending on the level of knowledge about the system [60]. Penetration testing is labor-intensive, requires a lot of expertise and does not necessarily lead to valuable results [13]. According to Schmittner et al [51], Pen testing is the best representation of a human attacker but has the drawback of requiring an almost finished system to perform tests on.

2.5.1.1 Penetration Testing Tools

In this section various software tools for penetration testing are listed. The mentioned tools are available mainly for personal computers. Depending on the target additional hardware

equipment might be required. For example, when access to an automotive Ethernet interface (100BASE-T1) is desired an adapter for the physical layer is needed. As most personal computers do not support cellular network standards, such as 3G and 4G, also in this case additional hardware is required. In case the software tool modifies the settings of the network interface card and the card is in use, it is recommended to use a dedicated one. Otherwise the connection for other applications might be lost.
Two categories of penetration testing tools can be found: the first category are operating systems distributed with already installed programs and the second category are single programs.

Tab. 2.1 lists operating system distributions for penetration testing. The operating systems listed provide a pre-installed set of tools for different purposes and are Linux-based. Additional tools are installed with Linux package managers. Kali is a Linux-based operating system available for x86 processors (32 bit and 64 bit) and for ARM processor architectures. The ARM support enables installations on devices such as a Raspberry Pi. The OS contains tools for information gathering, vulnerability analysis, wireless attacks, web applications, exploitation, stress testing, forensics, sniffing, spoofing, password attacks, maintaining access, reverse engineering, reporting and hardware hacking.
BackBox is a Ubuntu-based operating system available for x86 processors (32 bit and 64 bit). The OS contains tools for pen testing and security assessment from web application analysis to network analysis, stress tests, sniffing, vulnerability assessment, computer forensic analysis, automotive and exploitation. The automotive analysis category of BackBox includes can-utils.

Distribution	**OS**	**Platforms**	**License**	**System requirements**
Kali Linux	Linux	x86, x86-64, ARM	GPLv3	2048 MB RAM, 20 GB disk space
BackBox	Linux (Ubuntu)	x86, x86-64	mainly GPL	1024 MB RAM, 10 GB disk space

Table 2.1: Pen testing operating systems

Tab. 2.2 lists software tools that can be used to perform penetration tests. During penetration testing an attack is mimicked. The tester applies tools and techniques used for real-world attacks. Packet capturing tools such as Wireshark and tcpdump can be used for attack preparations. Traffic can be sniffed and analyzed to find vulnerabilities. Subsequently, an attack can be performed. There is a large number of tools for specific goals, e.g. `arpspoof` to manipulate a router's ARP table, but the focus of this thesis lies on broadly applicable tools and tool sets.
Wireshark is a network protocol analyzer with a graphical user interface. Packets can be captured live and saved to a file. Captured traffic of common network protocols such as TCP, UDP, TLS, and IP can be further analyzed. E.g. single TCP streams can be extracted and the payload can be displayed. Wireshark is available for a wide range of operating systems.

The program tcpdump is a Linux command-line tool to capture and analyze network traffic. Despite the TCP in the name, packets from various network protocols can be captured. Tools such as tcpdump and Wireshark use packet capture (pcap) application programming interfaces to capture low-level network traffic. Packets are directly captured at the network interface card circumventing the operating system's abstraction of connections. An example of a pcap API for Unix-like systems is Libpcap. For Windows it is Npcap. Tools such as Nmap can actively probe the network. Hosts and open TCP/UDP ports can be discovered.

The mentioned tools are mainly designed for networks based on the internet protocol suite (i.e. a set of communication protocols similar to the internet). Although Ethernet and IP-based networks are increasingly used in the automotive and off-highway industry, other bus systems such as CAN are common.
The Car Hacker's Handbook [41] provides a practical introduction to the topic of automotive penetration testing and lists common tools. The can-utils Linux tool set, for example, offers utilities for the Linux CAN subsystem (SocketCAN [72]) enabling a PC to gain access to a CAN network. Tools to display, record, generate and replay CAN messages ara available. CAN bus access via IP sockets is possible. Tools for bus measurement, bus testing, and utilities for the ISO-TP protocol are provided.
Caring Caribou is intended as a car security exploration tool with different modules. Modules supporting UDS, Universal Measurement and Calibration Protocol (XCP), CAN fuzz testing, dump CAN traffic, send CAN messages, and traffic monitoring are part of the installation.
PCAN-View is a program with a simple graphical user interface to transmit, record, and view CAN traffic. The Aircrak-ng suite offers tools to monitor and test WiFi networks. Packets transmitted wirelessly can be captured and exported. The tool offers functionality to crack WEP and WPA PSK encryption.
Scapy is a versatile Python program usable as command-line tool or as library. Network packets can be sent, sniffed or dissected. The tool further provides support for automotive specific protocols such as UDS, GMLAN, SOME/IP, ENET, OBD, CCP, ISO-TP and CAN. According to the Scapy documentation the automotive tools work best for Linux.

The above section presents a subset of available software tools. No claim to completeness is made. There is a high number of different tools for specialized goals. An exhaustive list is not in the scope of this thesis. A starting point, when searching for tools, is the Kali operating system documentation. The available programs are listed and categorized including a short description of each program [70].

Name	OS	License	Features
Wireshark	Windows, Linux, Mac OSX, Solaris, FreeBSD, NetBSD	open source, free, GPLv2	Network protocol analyzer, live packet capture
tcpdump	Linux	open source, free, BSD 3	packet analyzer, network traffic capture
Nmap ("Network Mapper")	Windows, Linux, Mac OSX	open source, free, GPLv2 with exceptions when embedding Nmap technology into proprietary software	Network discovery and security auditing
PCAN-View	Windows	proprietary, free	CAN monitor for viewing, transmitting, and recording CAN data traffic
can-utils	Linux	GPLv2, BSD-3	display, record, generate and replay CAN traffic, CAN access via IP sockets, CAN bus measurement and testing
Caring Caribou	Linux	open source, free, GPLv3	car security exploration tool
Aircrack-ng	Linux (primarily), Windows (limited functionality), Mac OSX, FreeBSD, OpenBSD, NetBSD, Solaris, eComStation 2	open source, GPLv2, BSD 3 Clause, OpenSSL	WiFi network security assessment
Scapy	Windows, Linux, Mac OSX, OpenBSD, SunOS/Solaris	open source, free, GPLv2	packet manipulation program and library

Table 2.2: Pen testing tools

3. Case Study: Onboard E/E System for an Off-Highway Vehicle

In this section an example use case is treated. At first the use case is described and subsequently analyzed in the context of cybersecurity. The use case is an electrical and electronic (E/E) system for an off-highway vehicle. The following process is influenced by the ISO/SAE 21434 draft [60] and follows steps from the concept and product development phase. The analysis does not conform to the standard but concepts and workflows are adopted.

3.1 Use Case

The example use case represents a common in-vehicle E/E network with different control units and communication interfaces. One subgroup of the E/E system consists of ECUs interconnected with a Controller Area Network (CAN) bus. The system controls the vehicle operation. The second subgroup is a human-machine interface (HMI) with two cameras attached. The cameras are placed at the front and at the back of the vehicle and the images are transmitted to the HMI. The HMI displays the images and allows the operator to overview the area. The cameras and the human-machine interface are connected to an automotive Ethernet switch over the 100BASE-T1 physical layer. The switch is additionally a small ECU managing the Ethernet packet distribution between the the two cameras and the human-machine interface.

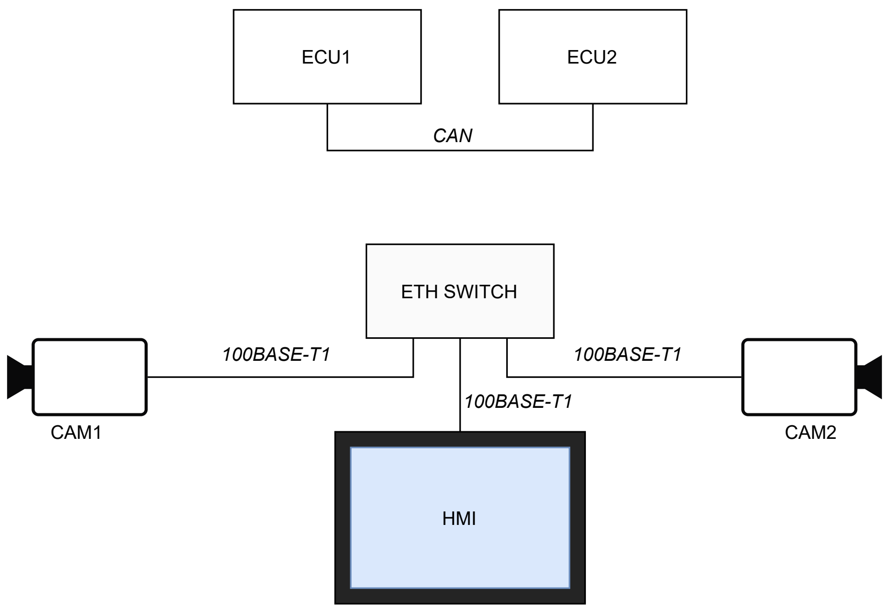

Figure 3.1: Preliminary vehicle E/E architecture

Fig. 3.1 illustrates the preliminary architecture of the vehicle's E/E system. The architecture is considered preliminary, as in the course of the following analysis the architecture is changed to mitigate threats to the system security. CAM1 and CAM2 are the cameras at the front and at the back of the vehicle. The HMI displays the images of the two cameras. The Ethernet switch (ETH SWITCH) enables switched Ethernet connections on the vehicle. The upper part of the architecture is separated from the lower part and consists of two electronic control units (ECU1 and ECU2) on the same CAN bus. ECUs on the CAN bus perform not further specified safety relevant actions controlling the physical actions of the system. The presented architecture serves as characteristic off-highway use case.

The goal of this thesis is to implement a secure external diagnosis for the system depicted in Fig. 3.1. Both subsystems shall be accessible over an external diagnostic connection. An external computer or diagnostic tool, denoted as *tester*, should have access to the HMI, the Ethernet switch ECU, and the two ECUs of the CAN bus. Diagnostic access allows the tester to read status information from the connected devices (e.g. error logs), to update their software, and to reconfigure settings.

3.2 TARA

In this section potential threats are analysed and the related risks are discussed.

Concerning the described use case it is assumed that the vehicular E/E system is physically protected by anti-tamper enclosures. The bus systems between the control units are also assumed to be inaccessible except when a connector is installed. It is important to note that these are strong assumptions for such a system, especially because physical-access oriented attacks are frequent in the automotive sector [57]. Nevertheless the assumptions are valid as attacks are often performed via already installed plugs and interfaces such as OBD, Bluetooth, WiFi, and cellular network (see [36], [43], [49], [57]). Protection mechanisms against

physical-access oriented attacks could influence the PCB (printed circuit board) design and the construction of the ECU housing. Measures to protect the microprocessors debug connector (e.g. locking after production) and to complicate access to the PCB's wires can be applied. It is also assumed that the software (i.e. bootloader, operating system, application) on the components is protected against tampering. The ECUs, the Ethernet switch and the HMI verify the software with protocols such as secure boot.
The assumptions made are summarized and listed:

A 1: The vehicular E/E system is physically protected by anti-tamper enclosures

A 2: The software on the components is protected against tampering

The topics covered by the assumptions are not treated in this thesis as this would go beyond the scope. In the following, mainly vulnerabilities of the communication system are discussed.

Enabling diagnostic access to both subsystems can be done by adding an automotive Ethernet and CAN connector located in the cabin. The connectors would allow direct access to both communication channels. Although the connectors are physically protected to some extend, if mounted inside the cabin, such an approach would expose the entire internal communication of the vehicle.

Advantages of the CAN bus are low costs of wiring, resilience to electromagnetic interference, self-diagnosis, and transmission error correction [57]. Current CAN bus systems are not designed with security in mind. There are no native security measures implemented in the CAN protocol. Without certain measures the CAN bus is vulnerable to attacks. Threats to the unprotected CAN protocol are masquerading attacks, eavesdropping, injection attacks, replay attacks, denial of service (DoS) attacks, and bus-off attacks [62].
The imitation of a legitimate node by an adversary is called masquerading attack. An eavesdropping attack takes place when an unauthorized party accesses the bus and manages to record the messages. In an injection attack an attacker transmits forged messages on the CAN bus. A replay attack happens when valid messages are resent by an adversary influencing the functioning of the system. In a bus-off attack the CAN transmit error counter is increased until the affected ECU disconnects from the bus. It is called a DoS attack, when high priority messages are repeatedly sent by an attacker blocking legitimate lower-priority messages.

Ethernet offers a high bandwidth and low costs of components due to the widespread use in information technology (IT) [32, p. 138]. Proven protocols on top of the physical layer such as IP, UDP, TCP etc. are supported by a wide range of consumer devices [32, p. 138]. Due to the exposure to electromagnetic interference, temperature, vibration, and moisture in vehicles, the plugs and cables of standard Ethernet are only suitable to a limited extend [32, p. 141]. Standard Ethernet (100BASE-TX) is, therefore, only used for diagnostic purposes. A cost-efficient automotive Ethernet physical layer is used for onboard communication. The physical layer is called BroadR-Reach and is standardized as 100BASE-T1 (IEEE 802.3bw). The standard supports 100Mb/s full-duplex operation over a single unshielded twisted-pair cable. Threats to automotive Ethernet networks are traffic confidentiality attacks, traffic

integrity attacks, DoS attacks and network access attacks [62].
In a network access attack an adversary gains access to the Ethernet network e.g. through unconnected switch ports or via remote access. Traffic confidentiality attacks allow the attacker to eavesdrop the traffic. Through actively sending frames, the network topology can be discovered or MAC tables can be manipulated (MAC flooding). Network access is required for such attacks. In traffic integrity attacks the network traffic is altered. ARP and DHCP poisoning attacks are attacks where the traffic is redirected to a malicious node. Replay attacks and session hijacking attacks are other examples for traffic integrity attacks. DoS attacks can be performed by physically damaging equipment or by overloading the system.

The internal communication channels CAN and Ethernet are used for diagnostic data and safety relevant communication. The safety relevant communication controls the physical actions of the machine. Hence, interference in the communication could cause physical damage to objects, animals or people.
Based on the mentioned possible attacks in this section, threat scenarios are elaborated. Threat scenarios are potential negative actions that lead to damage. The threat scenarios for both internal bus systems, the CAN and the Ethernet are summarized into threats for the *internal communication*. In the following, the identified threat scenarios for the internal communication are listed and enumerated:

TS 1: The communication could be interrupted (e.g. by a DoS attack or a bus-off attack)

TS 2: Messages of the communication could be forged

TS 3: Messages of the communication could be altered

TS 4: Messages of the communication could be resent (e.g. replay attack)

TS 5: The traffic of the communication could be eavesdropped

TS 6: An attacker could mimic a legitimate communication node (e.g. perform a masquerading attacks)

Access to the CAN and Ethernet buses is possible with readily available equipment, such as a laptop with Ethernet and CAN adapter. A relatively low level of expertise of the attacker is required. Hence, attacks on the internal communications are considered to be highly feasible. Countermeasures to improve the cybersecurity of the system and to reduce any potential risks are required.

3.3 Security Concept

Here follows the specification of cybersecurity goals to mitigate the risks from the analyzed threats.

The threats, specified by the threat scenarios (TS 1-5) concerning the internal communication of the system, are feasible only with direct access to the internal communication systems. The preliminary approach to enable diagnostic access by adding connectors to the internal bus systems is therefore discarded. Not only diagnostic traffic could be influenced by direct access, but also security relevant communications can be interfered by an attacker. The risk resulting from the identified threats is considered too high. Therefore, the first security goal is defined as:

Goal 1.1: The internal communication shall be protected from direct access

Goal 1.1 addresses the threat scenarios **TS 1** to **TS 6** for the internal communication. To protect the internal CAN and Ethernet communication a security gateway is added between the diagnostic interface and the internal buses. **Goal 1.1** is refined into the following security requirement:

SeqReq 1.1: A security gateway shall protect the internal communication from direct access

The gateway is the application of the firewall security pattern [61]. A firewall is placed between an untrusted external network and an internal network with weak security protection mechanisms. In the presented use case the untrusted external network is the diagnostic Ethernet network. The internal networks are the CAN and automotive Ethernet network. Only diagnostic messages are forwarded by the gateway. Diagnostic access to ECU1, ECU2, and the HMI is routed through the gateway. Fig. 3.2 presents the enhanced architecture with the additional gateway. The gateway device also contains an automotive Ethernet switch allowing the HMI to communicate with the two cameras (CAM1 and CAM2).

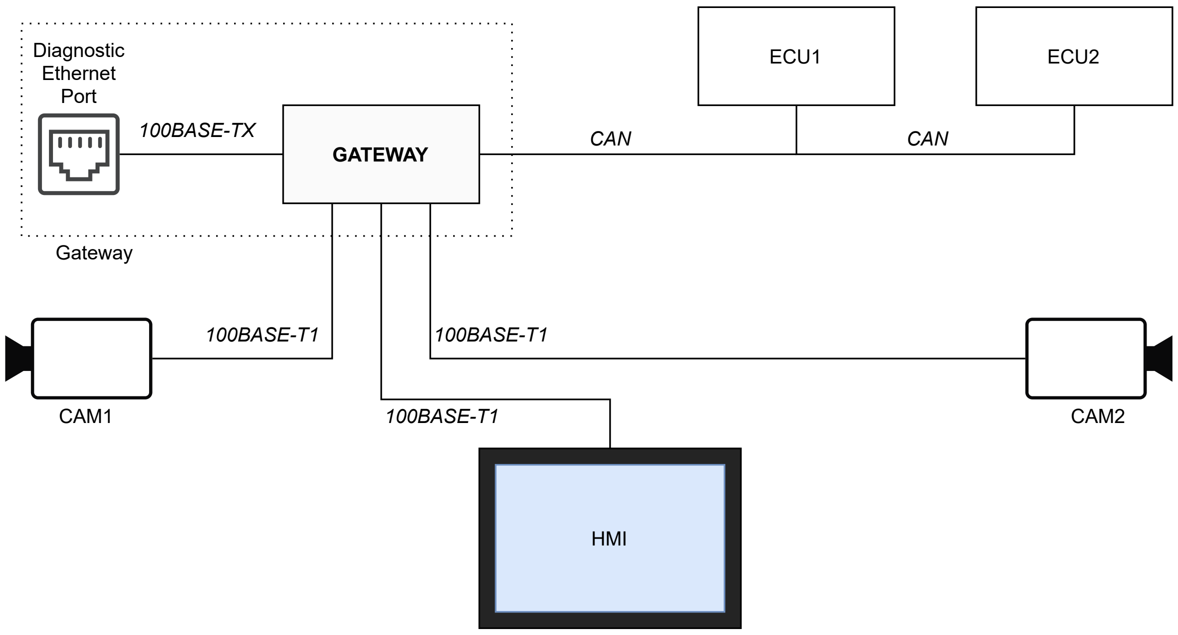

Figure 3.2: Vehicle E/E architecture with gateway ECU

With the placement of the gateway between the networks and the given assumptions, the internal vehicle buses are considered to be adequately protected. A still unresolved point is the security of the communication between the gateway and the tester.

To facilitate the architectural description and to select suitable protocols, a layered network architecture model is introduced. The five-layered Internet Protocol Stack [29, p. 49-52] is chosen for the Ethernet-based networks of the use case. The lowest abstraction layer is the physical layer, followed by the link layer and the network, transport and application layers. Ethernet standards, such as 100BASE-TX and 100BASE-T1, are part of the physical layer. The media access control (MAC) protocol and the frame format defined by the IEEE 802.3 Ethernet standard, specify the link layer.
On the network layer of the gateway's diagnostic port the Internet Protocol (IP) is selected. The Internet Protocol enables routing, forwarding, and fragmentation of packets. Furthermore a network layer address, the IP address, is introduced. A connection establishment with the gateway is now possible, using the IP address, which is independent from the MAC address. The MAC address is unique for each NIC (network interface card) and changes for every gateway hardware. The IP address can be reconfigured by the user and is independent from the NIC. Introducing the Internet Protocol enables the tester to establish a connection over a network to a host. Close proximity of the tester to the vehicle is no longer required. The vehicle's diagnostic port can be connected to a larger network making remote diagnosis possible. Such improvements in connectivity cause challenges for cybersecurity. No assumptions about the topology of the network between the gateway and the tester can be made. There could be other switches and routers in between. Furthermore wireless connections are also possible. The channel has to be assumed to be public.

The diagnostic traffic contains status information, software updates, and commands to reconfigure settings. Access to standard Ethernet or WiFi networks is feasible with readily available equipment. Equipment, such as a laptop with software tools (see section 2.5.1.1), can be used to identify vulnerabilities and to mount an attack. Information about the used protocols is publicly available for free (e.g. RFCs). To initially perform attacks on the system an experienced technician or engineer, knowing the protocols, is needed. The lack of protection mechanisms makes weaponizing an exploit ("take an exploit and make it easy to execute" [41, p. 193]) feasible. After weaponization, a proficient attacker or even a layman could pull off an exploit.
Eavesdropping attacks violate the confidentiality and privacy property of the transmitted data. An adversary can manipulate packets leading to a loss of message integrity. An attacker can also mimic a tester device and execute unauthorized commands. Overall attacks on the unsecured diagnostic connection are estimated to be highly feasible with minimal effort. The installation of the security gateway protects the internal bus systems from direct physical access. The protection of the diagnostic messages is an open point which is addressed in the following. The threat scenarios identified in section 3.2 are applied to the external diagnostic connection. **TS 1** to **TS 6** are applicable also for the external connection. The fact that the external communication is for diagnostics only, reduces the negative impact of damage scenarios potentially caused by threat scenarios.
The following cybersecurity goals regarding the diagnostic messages are identified:

Goal 2.1: The diagnostic message's integrity shall be protected against spoofing

Goal 2.2: The diagnostic message's confidentiality shall be protected against eavesdropping

Goal 2.3: The diagnostic message shall be protected against replay attacks

Goal 2.4: The tester device shall be authenticated

Goal 2.5: The gateway device shall be authenticated

The listed additional cybersecurity goals are derived for the threat scenarios **TS 2** to **TS 6** in the context of the external communication. **TS 1** mentions the possibility of an interruption of the external connection. Connection interruptions in public communication channels are difficult to eliminate. Additionally, no detailed assumptions about the public channel can be made. For diagnostic communication the interruption of a connection is assumed to be acceptable in this context. Therefore, no cybersecurity goals are derived for **TS 1** .
Goal 2.1 addresses the threat scenarios **TS 2** and **TS 3**. **Goal 2.3** addresses threat scenario **TS 4**. **TS 5** is addressed by **Goal 2.2**. **Goal 2.4** and **Goal 2.5** address the threat scenario **TS 6**.

The cybersecurity goals **Goal 2.1** to **Goal 2.5** are further refined by specifying cybersecurity requirements. The requirements are allocated to the gateway ECU and the tester device. The following cybersecurity requirements are derived:

SeqReq 2.1: The diagnostic messages shall contain a message authentication code (gateway and tester)

SeqReq 2.2: The message authentication code of the diagnostic messages shall be verified (gateway and tester)

SeqReq 2.3: The diagnostic messages shall be encrypted (gateway and tester)

SeqReq 2.4: The communication protocol shall provide countermeasures against replay attacks (gateway and tester)

SeqReq 2.5: The tester shall provide a digital certificate for authentication (tester)

SeqReq 2.6: The gateway shall provide a digital certificate for authentication (gateway)

The security requirements **SeqReq 2.1** and **SeqReq 2.2** refine the cybersecurity goal **Goal 2.1**. **Goal 2.2** is refined by **SeqReq 2.3**. **SeqReq 2.4** refines **Goal 2.4**. **SeqReq 2.5** and **SeqReq 2.6** concretize the respective security goals **Goal 2.4** and **Goal 2.5**. Regarding the certificates, it is assumed that certificate authorities the system relies on are already available and appropriately managed. The management of such certificates and its associated public key infrastructure (PKI) are out of scope for this work. Both, the tester and the gateway shall authenticate themselves to avoid any man-in-the-middle attacks (MITM).

3.4 System Architectural Design

The development of an architectural design is the first step of the product development phase. During this phase adequate solutions for the above-specified requirements are found.

The hardware components of the system are off-the-shelf components. The hardware of the gateway ECU is described in further detail in chapter 4. The gateway ECU is a TTConnect 616 from the manufacturer TTControl. The device offers the required CAN, 100BASE-TX and 100BASE-T1 interfaces. An Ethernet switch is integrated in the ECU. The remaining ECUs (ECU1 and ECU2) are not further specified. In this thesis only the gateway ECU and its interfaces are of interest (denoted by a dashed square in Fig. 3.2). The HMI and the cameras are also seen as generic components and not further specified.
The tester device is assumed to be an adequate communication counterpart with the required functionality already implemented. In this thesis, the software components of the gateway ECU are designed and developed.

The diagnostic protocol shall provide functionality to read status information from the connected devices (e.g. error logs), to update firmware, and to reconfigure settings. The Unified Diagnostic Services (UDS) protocol provides such functionality. The UDS standard describes a diagnostic protocol in the application layer independent from the underlying layers [32, chapter 5]. UDS is standardized in ISO 14229. It is decided to select UDS as diagnostic protocol because its layered design allows integration on different bus systems. It is assumed that the ECUs on the internal CAN bus are running a UDS on CAN stack (UDS on CAN, ISO 14229-3). The HMI also supports UDS but over automotive Ethernet. UDS via IP-based networks is enabled by the Diagnostics over Internet Protocol (DoIP) standardized by ISO (ISO 13400).

DoIP lacks security mechanisms to ensure authenticity and integrity of the transmitted data [30]. According to Kleberger et al [30] the implementation of IPsec results in the most desirable security architecture. IPsec (Internet Protocol Security) enables secure connections between devices. The protocol resides at the network layer. TLS in contrast is located between the transport layer and the application layer. IPsec provides authentication and encryption of packets.
A drawback of IPsec is the lack of freely available open-source implementations for embedded systems. A partial implementation was done by Scheurer and Schild as a diploma work [5]. The implementation lacks support of modern hash and encryption algorithms. Only insecure hash algorithms, such as MD-5 and SHA-1, are integrated. Furthermore, the authors state that the implementation is not market-ready in a sense of robustness, flexibility, and ease of use. The project was released in 2003 and no further development was done since then. On the contrary, numerous free open-source implementations are available for the TLS protocol. OpenSSL, GnuTLS, and mbed TLS are examples.
Therefore TLS is selected to secure the diagnostic communication between tester and gateway. TLS protects the integrity and confidentiality of messages. Messages are encrypted and a message authentication code is appended. The communication parties are authenticated using public-key cryptography. The underlying Transmission Control Protocol (TCP) ensures reliable transport. A limitation of TLS is that the broadcast functionality of UDS to detect DoIP-capable devices in the network is not supported [30].

Fig. 3.3 illustrates the design of the gateway software for the diagnostic port. For each of the five layers of the Internet Protocol Stack a protocol is selected. Standard Ethernet defines the physical and link layer. The IP protocol is selected for networking. TLS requires TCP as transport protocol. The application layer consists of the UDS diagnostic protocol together with DoIP and the TLS security protocol. The implementation of the gateway's software is described in chapter 5.

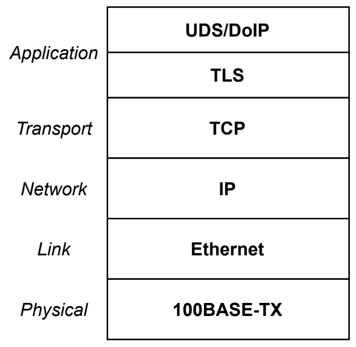

Figure 3.3: Gateway software architecture

In the next chapter the ECU hardware and the microcontroller's security features will be described.

4. Gateway ECU Hardware

The following chapter describes the electronic control unit (ECU) used for the thesis project. The focus lies on security relevant interfaces and components. Features which are not of interest for the thesis, such as digital inputs and outputs, are not described.
The gateway used for the project is an ECU developed for vehicles and machines in rugged operating environments. TTControl's off-highway ECU TTConnect 616 supports connectivity on several interfaces [66]. The device is intended for the use as switch or gateway in modern vehicles. The TTConnect 616 ECU is delivered in an aluminum housing with dimensions of 147 x 92 x 38 mm. Fig. 4.1 shows an image of the ECU.

Figure 4.1: TTControl TTConnect 616 (image from ttcontrol.com)

The project's ECU is a revised variant of the TTConnect 616 [65]. The revised variant has a newer generation microcontroller and updated Ethernet switch hardware. An overview of the ECU's features is given in Tab. 4.1.
As noted in Tab. 4.1, some interfaces are only supported as hardware. Hardware-only supported means that the transceivers are mounted on the PCB, but no software support is available to the date the project was done.

The ECU's main components are a microcontroller and two automotive Ethernet switches. The automotive microcontroller is used for two purposes: to configure components of the ECU and to run an application. E.g. the two switches and the transceivers for the CAN bus and Ethernet are configured by the microcontroller.
The switches are configured to operate in a chained mode where two 5-port switches are used to create one 8-port switch. One port on each switch is used to interconnect the two of them. The IEEE 802.3-compliant switch device [40] is called a MAC (media access control) component allowing different PHYs (physical layers). The PHYs attached to switches on this ECU are six BroadR-Reach (100BASE-T1) transceivers and a standard Ethernet (100BASE-TX) transceiver. MAC-to-MAC and MAC-to-PHY communication is realized with the media-

Microcontroller [59]	• Infineon Aurix TriCore TC377TP • 32-bit • 3 cores running at 300 MHz • 6 MB flash • 1136 kB SRAM • Hardware Security Module (HSM)
Ethernet Switch	2 x SJA1105Q (5 port switch)
Interfaces	• 6 x 100BASE-T1 (BroadR-Reach) • 1 x 100BASE-TX • 6 x CAN • 1 x LIN* • 1 x FlexRay*
Inputs/Outputs	• 3 x digital input or analog input* • 2 x digital output (high-side switch)*

Table 4.1: Gateway ECU features overview (*hardware support only)

independent interface (MII).
100BASE-T1 is a 100 Mbit/s Ethernet PHY standard for automotive applications. The PHY realizes full-duplex communication via a single unshielded twisted-pair cable. The standard aims to reduce cabling cost and weight [32]. Furthermore 100BASE-T1 components are adapted to automotive environments regarding electromagnetic compatibility (EMC), temperatures, vibrations and humidity exposure [32]. 100BASE-T1 is standardized in IEEE 802.3bw.
100BASE-TX is a popular standard among consumer devices such as PCs. It realizes 100 Mbit/s full-duplex communication via two twisted-pair cables. 100BASE-TX is standardized in IEEE 802.3u. Those standard Ethernet components are not considered suitable for automotive purposes but adequate for production or repair-shop environments [32]. A benefit of the widely supported standard is the availability of cheap equipment required to access vehicles in the aforementioned environments.
The eighth switch port is connected to the microcontroller. There is also an additional Serial Peripheral Interface (SPI) bus connection between the microcontroller and the switches. The SPI bus is used to enable and configure the switches.
Fig. 4.2 shows a block diagram of the ECU's main components. The two chained switches are illustrated. The diagram shows only interfaces that are both, supported in software and hardware and therefore fully available. The six CAN transceivers are directly connected to the microcontroller as the Aurix already supports the CAN protocol. The Ethernet port of the Aurix is connected to the Ethernet switch. This enables the communication from the microcontroller to each of the other ports and vice versa. With the respective switch configuration it is possible to disable or enable switch ports and to separate traffic going through the switches.
In the following sections more insight into the architecture of the Ethernet switches and the microcontroller is given.

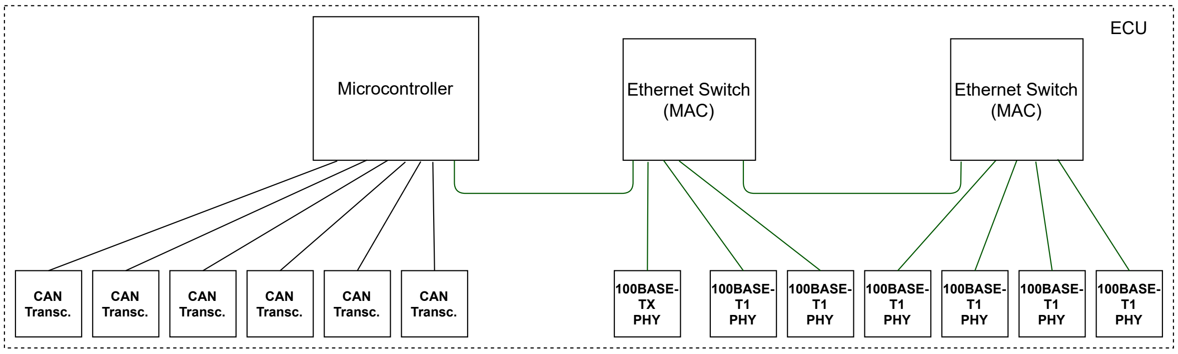

Figure 4.2: Gateway ECU block diagram (simplified)

4.1 NXP Ethernet Switch

The ECU has two cascaded SJA1105Q automotive Ethernet switches to create one 8-port switching unit. The automotive switches are IEEE 802.3 compliant and support IEEE 802.1Q frame tagging and priority-based QoS handling [46]. Furthermore, features required for Time-Sensitive Networking (TSN) and Audio Video Bridging (AVB) are supported [46]. The port's data rates range from 10 to 1000 Mbit/s [46].
For the current variant of the ECU TSN and AVB are not enabled. The switches are handling best effort traffic only. The 100BASE-TX and 100BASE-T1 PHYs support a maximum data rate of 100Mbit/s. Therefore, the switch ports are limited to 100Mbit/s. To avoid a bottleneck in the interconnection of the two switches this connection is operating with 1000Mbit/s. IEEE 802.1Q frame tagging is enabled to separate network traffic on layer 2. Configuration of virtual local area networks (VLANs) is implemented.
The switch also supports statistics about frames and buffer load [46]. Such diagnostic information is not only beneficial during development but also allows error detection and monitoring during operation.

4.2 Infineon Aurix TriCore Microcontroller

This section introduces the microcontroller integrated in the ECU. The microcontroller is highly relevant for the overall system security as it is the ECU's core component controlling all the peripheral components and the communication interfaces. The microcontroller of the ECU is an Infineon Aurix TriCore TC377TP. This second generation Aurix is a state-of-the-art 32-bit automotive and industrial microcontroller. The chip offers three cores running at 300MHz, 6 MB of flash and 1.1 MB SRAM. Supported interfaces are CAN FD (CAN Flexible Data-Rate), FlexRay, LIN, SPI, I^2C, SENT, PSI5 and MSC. To improve safety, two out of the three cores are running in lockstep mode. Lockstep mode is when a core has an additional lockstep core executing the same instructions in a time-shifted manner. The result of the main core and the lockstep core is compared subsequently. [59]
The microcontroller chip integrates an additional security feature, namely a dedicated security coprocessor. The coprocessor is called Hardware Security Module (HSM) and is full EVITA compliant [59]. Further details about the HSM are given in the following section.

4.3 Hardware Security Module (HSM)

In this section the architecture of the HSM integrated in the Infineon Aurix TriCore microprocessor is described. The Aurix' HSM is categorized as EVITA full HSM. The architecture of an EVITA full HSM is described in section 2.3.

4.3.1 Aurix TriCore HSM

The Infineon Aurix TriCore microcontroller, used in the ECU, has an integrated HSM. In contrast to the previous generation of TriCore microcontrollers (TC2xx), the 2nd generation Aurix TriCore (TC3xx) has an EVITA full HSM.
Not all building blocks of the 2nd generation Aurix HSM are identical to the EVITA full HSM specification described in section 2.3. Instead of the Whirlpool hash function the SHA256/224 has been implemented in hardware. Infineon points out that SHA-2 algorithms have established themselves as cryptographic hash engines on the automotive market [54, p.4]. As counter block not sixteen 64-bit counters are available but two 16-bit timers [53]. Despite the architectural differences, Infineon states the Aurix TriCore HSM being an EVITA full HSM [59, p. 87]. The following Tab. 4.2 lists the building blocks of the Aurix 2nd generation HSM. Blocks differing from the EVITA specification are marked with a star (*).

Full EVITA HSM building block (Fig. 2.1)	**Aurix 2G HSM module**
Secure CPU	ARM Cortex M3 (100MHz)
Internal RAM	96kB of RAM
Internal NVM	640kB PFlash and AES module's private key storage
Asymmetric Crypto Engine	public key cryptography (PKC) engine
Symmetric Crypto Engine	AES-128 module
TRNG/PRNG	TRNG, AES- and hash module can be used for PRNGs
Hash Engine	* Hash module supporting MD-5, SHA-1, SHA-224, SHA-256
Counters	* Timer unit (two 16-bit timers)
Hardware Interface	Bridge module

Table 4.2: Aurix 2G (2nd generation) EVITA full HSM based on [53] and [22] (* blocks differing from EVITA full HSM specification)

The EVITA application core is named host system by Infineon and consists of several peripherals, RAM, Flash memory, sensors and three TriCore CPUs. In the following sections, the different modules and features of the Aurix 2G HSM are described in more detail. The technical specifications are taken from the Automotive Cyber Security Compendium for the Infineon Microcontroller AURIX [53].

4.3.1.1 Secure CPU

The secure CPU is an ARM Cortex M3-based 32-bit coprocessor. A frequency of up to 100 MHz is supported. The Cortex-M3 is a low-power processor with one core [17]. Debugging is possible during development and can be deactivated for shipment.

4.3.1.2 Internal RAM and NVM

The HSM has 96kB local RAM and local boot ROM protected from access by the host system. The boot ROM contains code and read-only data. The contents of the boot ROM are necessary for the boot-up of the HSM. The HSM program is stored in the microprocessor's PFlash and DFlash. Therefore, it is crucial to store the HSM data and code to separate regions. Furthermore, those memory regions are required to be locked and protected before the ECUs are shipped.

4.3.1.3 Asymmetric Crypto Engine

The HSM's public key cryptography (PKC) module provides operations required for asymmetric cryptography. Modular and non-modular mathematical operations on integers and polynomials are supported. The PKC module can perform multiplications, modular addition and subtraction, modular multiplication, modular inversion and division as well as modular exponentiation. Integers and binary polynomials up to 256 bit length are supported. Algorithms on specific elliptic curves are also provided: addition of two points, doubling of a point and scalar multiplication. The following elliptic curves are supported [69]:

- NIST curves [28]
- Brainpool curves [19]
- Ed25519 curve [39]

Ed25519 is the twisted Edwards curve equivalent to the Montgomery curve Curve25519 [39, p. 5].
The hardware module of the HSM does not support complete signature generation or shared secret calculations based on elliptic curve cryptography (ECC). Such algorithms have to be implemented in software with hardware support for single operations (i.e. point addition, point doubling, and scalar multiplication). More details about algorithm implementations are provided in chapter 5.5.

4.3.1.4 Symmetric Crypto Engine

As symmetric crypto engine, the HSM has an Advanced Encryption Standard (AES) module. The module supports encryption and decryption of 128-bit-key AES (i.e. AES-128 [3]). The module offers internal write-only storage for up to 8 keys. After saving, the keys are not readable anymore. To enable encryption/decryption of data larger than the AES block size (128 bit) different modes of operation are available. The following modes of operation are supported:

- ECB (electronic codebook) [4]
- CBC (cipher block chaining) [4]
- CTR (counter) [4]
- OFB (output feedback) [4]
- CFB (cipher feedback) [4]
- GCM (Galois/Counter Mode) [9]
- XTS (XEX Tweakable Block Cipher with Ciphertext Stealing) [20], [14]

The ECB, CBC, CTR, OFB, CFB, and XTS mode provide confidentiality. The GCM mode provides confidentiality and authenticity of the encrypted data. Modes providing confidentiality do ensure that the plaintext is converted into unreadable ciphertext but do not ensure that the ciphertext is not altered. Additional mechanisms, such as message authentication codes (MACs), are required to detect modifications of the ciphertext.

4.3.1.5 TRNG/PRNG

The HSM has a built-in true random number generator (TRNG) module. A non-deterministic noise source is digitized and post-processed. The entropy of the generated bit stream is then checked. In case the entropy is low, a warning will be raised. The random number generator meets the requirements for the functionality class PTG.2 by the German BSI (Bundesamt für Sicherheit in der Informationstechnik) [69]. The random number generator (RNG) class PTG.2 describes a physical RNG with internal tests detecting a total failure of the entropy source and non-tolerable statistical defects [24, p. 7]. Furthermore, a statistical model of the entropy source is implemented and statistical tests on the output are performed [24, p. 7-8]. The output of such a TRNG, or physical RNG, can be used as input for a pseudorandom number generator (PRNG), or deterministic RNG. The process of using such an entropy source to initialize the state of a PRNG is called seeding [24, p. 15]. RNGs using an entropy input and cryptographically post-processing it are categorized as functionality class PTG.3, the strongest class defined by the BSI [24, p. 79]. The AES-128 module and the hash module of the HSM can be used to build such pseudorandom number generator (PRNG) using a seed from the HSM's TRNG. Hash- and block cipher-based PRNGs are specified by NIST in the freely available special publication 800-90Ar1 [33].

4.3.1.6 Hash Engine

The HSM's hash module supports the following hash algorithms:

- MD-5
- SHA-1
- SHA-224 [34]

- SHA-256 [34]

The MD-5 and SHA-1 hash functions are not considered collision resistant [63, p. 18]. The term collision resistance describes that it is infeasible (computationally hard) to find two messages that generate the same hash output or to modify a message without changing the hash output [27, p. 126]. After thorough examination, these two hashes could be used for applications where collision resistance is not crucial [63, p. 18]. The German BSI recommends the usage of selected hash functions from the SHA-2 and SHA-3 family. SHA-256 is the only hash algorithm supported by the HSM, which is considered to be cryptographically strong according to the BSI [63, chapter 4].

4.3.1.7 Counters

The timer module of the HSM offers two independent 16-bit up-counting timers. The clock source and the clock prescaler are configurable.

4.3.1.8 Hardware Interface

The HSM's hardware interface to the host system is called bridge module. The bridge module controls all interactions between the HSM and the host system. Via the bridge module the HSM is able to trigger interrupts on the host system side and vice versa. The architecture allows full access to the host system by the HSM. The HSM has access to memory and peripherals, while the host system has only restricted access to the HSM. During operation the host system can only access dedicated communication registers. The communication registers consist of flags and 32-bit status values.

4.3.1.9 Additional Modules

The following two modules are not explicitly specified by the EVITA project (section 2.3) and are, therefore, only briefly described.

Watchdog Timer
To monitor the system the timer module features a 16-bit watchdog timer. The watchdog timer could be seen as additional counter in Tab. 4.2.

Debug Modules
The HSM's debug modules allow debugging of HSM code during development.

5. Gateway ECU Embedded Software

The previous chapter outlines the ECU and its project-relevant hardware components. This chapter describes the software components running on the ECU. The architectural aim of the developed software lies in encapsulating security-relevant computations in the HSM and making full use of the available cryptographic hardware modules. Fig. 5.1 illustrates the main components of the developed software solution. Green components (host application, HSM interface, and HSM application) are developed as part of this thesis project.

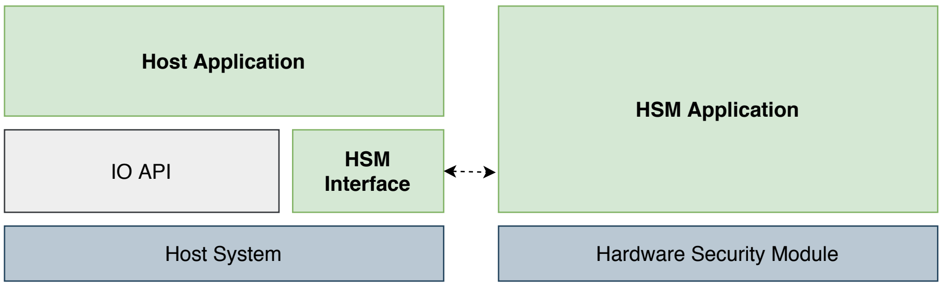

Figure 5.1: Main components

The host system and the Hardware Security Module (HSM) in Fig. 5.1 are the two hardware components executing the software. The HSM and the host system are two separate microprocessors interacting with each other. The code running on the HSM secure CPU is denoted HSM application. The HSM application provides cryptographic services (encryption, decryption etc.) to the host application. The host application accesses HSM functionalities via the HSM interface software component only.

The HSM interface manages the interconnection between the two microprocessors. The interface deploys tasks to the HSM and notifies the host application upon completion or when errors occur. In Fig. 5.1 the information exchange between the two software components is represented as the two-pointed arrow between them. The communication itself is done via shared RAM areas and communication registers.

The IO API is provided by the ECU's manufacturer. The author of the thesis was involved in the development of the IO API. The specific version used for the project is TTConnect 616C CAPI 1.0.0.1, the latest release. Interfaces such as CAN and Ethernet are accessed by the application using the IO API. Furthermore, the IO API provides general initialization routines for the ECU. Especially relevant for the project are API functions regarding the

Ethernet transceivers, BroadR-Reach transceivers, and the Ethernet switch.
The host application defines the behavior of the ECU. Different applications can be developed for various use cases. For example, an application could pack CAN messages into Ethernet frames and forward them to dedicated ports. In case encryption or digital signatures are required, the host application makes use of the HSM interface. The host application developed for the thesis is described in detail in section 5.2.

5.1 Software Implementation

The previously described software architecture is implemented in C programming language. The host application, the HSM interface, and the HSM application were developed during this thesis project. The Git version-control system is used to track changes in source code during software development. Two repositories, one for the HSM application and one for software, running on the host system, are created. The chosen source-code editor is Visual Studio Code by Microsoft.

Toolchain

The project's Infineon Aurix Tricore microcontroller (TC377TP) contains two different processor core architectures. The host system consists of three Tricore CPU cores and the HSM secure CPU has one ARM Cortex core. For each architecture a different compiler is required. To build the project for the Tricore, the HighTec TriCore Development Platform v4.9.3.0 is used. As a toolchain to build for the HSM's ARM architecture the TASKING VX-toolset for TriCore v6.2r2 is used. The licenses for the compilers are provided by TTControl.

Debugging

For debugging a professional debugger is used, namely the Lauterbach POWER DEBUG INTERFACE / USB 2 with DEBUG cable and TRACE32 PC software. Tricore and ARM debug licenses are provided by TTControl.

In the following, the software components illustrated in Fig. 5.1 are outlined.

5.2 Host Application

The host application of the thesis aims at providing a demonstrator for a secure communication protocol. In the final product the host application provides secured diagnostic services. The UDS server is listening for incoming diagnostic messages and responds to them. The diagnostic traffic is encrypted and protected against tampering by a secure communication protocol. Due to the limited time frame, the UDS server application is replaced by an already available implementation of an HTTP server.

The host application runs on the host system using functions provided by the IO API and the HSM interface. The IO API provides an implementation of the Ethernet protocol. Physical and link layer are, therefore, already available on the ECU. The implementations of TCP and IP are provided by the already integrated TCP/IP stack. The integrated stack is called lwIP [71]. LwIP is an open-source project designed for embedded systems with a focus on reduced resource usage. The version of the integrated lwIP stack is 2.1.2.
For the use case, TLS is chosen to secure the communication between a diagnostic device (e.g. PC) and the ECU. The lwIP TCP/IP stack supports the integration of a third party TLS layer. For mbed TLS an example port to lwIP is already available. Mbed TLS focuses on a small footprint and portable code [73]. The API is well documented and the code is readable. Hardware support for certain cryptographic algorithms can be integrated. TLS is supported up to version 1.2. Due to the ease of integration, the license (Apache-2.0), and the available examples, mbed TLS is selected as TLS implementation and integrated in the host application. The version of the used mbed TLS is 2.13.1.
On application layer UDS was selected during the case study in section 3.4. Integrating UDS on the device would go beyond the scope of this master thesis. Therefore, the HTTP server of the TCP/IP stack is enabled, providing an example web page. Using an HTTP application with TLS results in an HTTPS (Hypertext Transfer Protocol *Secure*) server. The HTTPS server application makes use of the identical underlying protocols as the secured version of UDS with DoIP (see Fig. 3.3). Consequently, this server application is used to test and verify the functionality of the underlying protocols (TLS, TCP, IP, and Ethernet). Simple tools can be used to communicate with the ECU. A browser on the development PC, for example, is used to open the web page of the server. The traffic is observed with the Wireshark tool.

According to the German BSI only TLS version 1.2 and 1.3 are recommended [64, chapter 3.2]. The highest supported version by mbed TLS is 1.2. Therefore, TLS 1.2 is selected for the project. The configuration of mbed TLS allows to select the supported ciphersuites. Usually, when implementing a web server it is required to support a wide range of ciphersuites in order to avoid compatibility issues. In this case, compatibility is no concern as the client (tester device) is developed especially for the gateway ECU. Accordingly, the strongest ciphersuite supported by the HSM hardware acceleration is selected. The ciphersuite with the IANA name [58] `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` is selected. The naming scheme for the `TLS` ciphersuite is the following: `ECDHE` indicates that the elliptic-curve Diffie–Hellman (Ephemeral) is used as key exchange algorithm, while `ECDSA` indicates the client and server authentication mechanism which is elliptic curve digital signature algorithm (ECDSA). `AES_128_GCM` indicates that AES with 128 bit key length in Galois/Counter Mode (GCM) mode is used for encryption, and `SHA256` indicates that the SHA-256 hash algorithm is used. The ciphersuite is specified by IETF in RFC 5289 [12]. The authenticated encryption with additional data algorithm (`AES_128_GCM`) provides the message encryption and the message authentication code (MAC). The algorithms of the selected ciphersuite are also compatible with the successor TLS 1.3.

Mbed TLS offers the possibility to substitute provided cryptographic primitives with alternative implementations. Functions provided by the HSM Interface (section 5.4) are integrated in the TLS implementation. The entropy source of the HSM is used (`HsmIfTrngGenerate`). The hash generation is moved to the HSM secure core by integrating the `HsmIfSha256Process` function. The message encryption and the generation of the MAC are also performed by the HSM (`HsmIfAesCryptGCM`). As well as the message decryption and the MAC verification are done by the HSM. Computations for the key exchange (`HsmIfEcdhGenPublic, HsmIfEcdhComputeShared`), the signature generation (`HsmIfEcdsaSign`) and verification (`HsmIfEcdsaVerify`) are moved to the HSM core. The deterministic random bit generator of the TLS implementation makes use of the AES-ECB block decryption and encryption algorithms. The AES-ECB computation is also moved to the HSM. All cryptographic primitives of the TLS implementation are externally computed by the HSM secure core.

Fig. 5.2 presents the implemented architecture of the host application. Compared to the design in section 3.4 (Fig. 3.3), the UDS/DoIP segment is substituted with the HTTP server. Except for the TLS, all parts of the TCP/IP stack are provided by the lwIP implementation. The TCP/IP stack is integrated on top of the IO API providing the Ethernet functionality. The TLS implementation makes use of the cryptographic functions provided by the HSM interface.

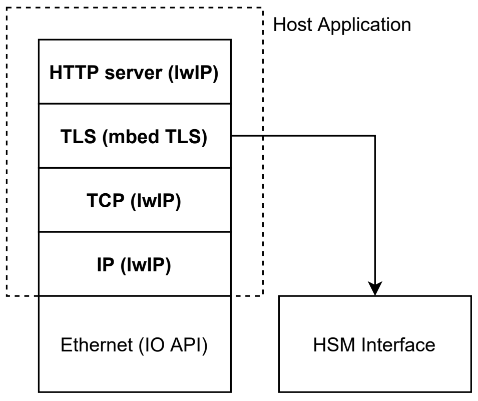

Figure 5.2: Host application architecture (UDS replaced by HTTP)

5.3 IO API

The IO API offers an interface for application development in the C programming language. The library is provided by TTControl and the complete name with version is TTConnect 616C CAPI 1.0.0.1. This release supports the CAN interface, the Ethernet interface and Ethernet switch and transceiver configuration. Different Buffers, baudrate and filter configurations are available for CAN interfaces. Sending and receiving IEEE 802.1Q tagged Ethernet frames is supported. The Ethernet switch interface offers functions to read certain frame error counters and the number of sent frames. This version of the IO API supports only static switch configuration. The static configuration is stored in the ECU's flash memory and loaded during the initialization of the switch. VLAN memberships for each port can be set

by a tool that generates the static configuration. The tool creates files to include in the C application project which then are compiled and flashed to the ECU's memory. Additionally, the tool enables the user to configure the BroadR-Reach ports either in master or slave mode. Dynamic configuration would allow the ECU application to change the switch configuration dynamically during runtime. For example, the VLAN memberships of ports could be changed depending on inputs and/or messages received.
The IO API offers functions to initialize the ECU's Ethernet transceiver and the BroadR-Reach transceivers. Using the configuration files generated by the tool, the BroadR-Reach transceivers can be initialized as master or slave. Additional transceiver status information can be obtained via the API. Disabling an Ethernet or BroadR-Reach port is done by omitting the call of the initialization function of the transceiver. The IO API is used by the host application to transmit and receive messages via the Ethernet and CAN interface and to initialize and configure the transceivers and Ethernet switch. The development of the IO API is, however, not part of this thesis.

5.4 HSM Interface

The HSM interface software component manages the interaction between the host system and the HSM. The exchange of data between the two microprocessors is done via shared RAM areas and communication registers. While the communication registers are used to exchange small amounts of data (32 bits) in both directions and to trigger interrupts on HSM side, the shared RAM area is used to exchange larger amounts of data. The task of the HSM interface software component is mainly to transfer parameters and to trigger the execution of cryptographic algorithms on the HSM. Additionally, the interface software initializes the communication between the two microprocessors. During the initialization, a check is performed to ensure the HSM booted correctly and is ready for tasks. Furthermore, the address of the memory interface is transmitted to the HSM using the host-to-HSM status register.
The memory interface is a structure for information exchange stored in the host system's RAM. The structure variable holds pointers, a status value, as well as 32-bit and 64 values. The pointers are memory addresses of buffers which are either used as input or output for the HSM. Each buffer has an associated buffer length variable. The structure holds one status variable which is used by the HSM to communicate error messages and general status information.
If not an array or buffer is required, the memory interface structure also offers 32-values and 64-bit values. The structure offers in total five buffer address entries with associated buffer length variables, four 32-bit variables, two 64-bit variables, and one status variable. 76 bytes of RAM memory are occupied by the memory interface structure. The structure variable is instantiate by the HSM interface in the host system's RAM. During the initialization function, the memory address of the instantiated variable is passed to the HSM. All subsequent HSM tasks exchange information and data with the host system using exactly this memory interface.
Fig. 5.3 presents the memory interface and communication registers used by the HSM inter-

face and HSM application. The figure does only include only the used registers. Registers which can be used to trigger interrupts on the host system side are excluded as they are not used in this software implementation. Host-to-HSM registers can only be written by the host system. HSM-to-host registers can only be written by the HSM.

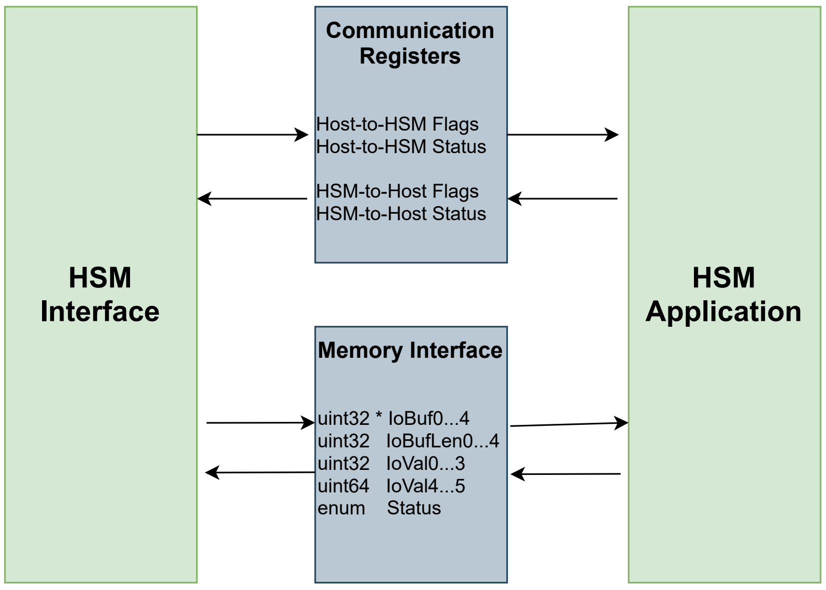

Figure 5.3: Host system/HSM interface

5.4.1 Task Deployment

This section describes the general task deployment process. No specific procedures for the single cryptographic algorithms are outlined, but all tasks share a temporal execution scheme. The process is started by the host application calling an HSM interface function. The interface function then sets the variables of the memory interface structure depending on the desired HSM algorithm. Subsequently, the host-to-hsm flags register is set to trigger the execution on the HSM. Write access by the host system to host-to-hsm flags register triggers an interrupt on the HSM. The HSM application checks the parameters of the memory interface and starts to compute the desired cryptographic algorithm. Finally, the results are either written to the provided buffer addresses or to individual memory interface values. In case an error occurs, the memory interface's status variable is set to pre-defined error code. The host application polls the HSM-to-host flags register until the task is completed. After completion, the HSM-to-host status register is read and compared with an expected value to diagnose bridge module errors. Both, the HSM-to-host status register and the memory interface's status variable, are used to signal errors or warnings to the host system. The register value is used for errors occurring in communication between the host system CPU and the HSM secure CPU. The status variable of the memory interface is used to communicate errors and warnings arising during the algorithm execution. Incorrect parameters passed, null pointers, or a failed digital signature validation are examples.

5.4.2 HSM Interface Functions

In this section the implemented functions of the HSM interface are listed and described. The functions are called by the host application to initialize the HSM and to deploy tasks to the HSM's secure coprocessor. The provided C functions are categorized by algorithm (AES, ECDSA etc.). In the following, each category is described and the functions are listed. For each category one header (h) file is provided which can be included by the host application. The function to initialize the HSM interface is called `HsmIfInit` and needs to be called before any other function. Each function returns error codes in case a failure has occurred. The return values and error codes are not described in detail as the focus is on the functionality of the interface.

5.4.2.1 AES

This paragraph outlines the interface functions based on the AES-128 block cipher. The functions listed launch tasks on the HSM making use of the symmetric cryptography engine described in section 4.3.1.4. The AES module offers a key storage and different modes of operation. Supported modes of operation are ECB (electronic codebook), CBC (cipher block chaining), CTR (counter) [4], and GCM (Galois/Counter Mode) [9]. Tab. 5.1 lists the provided C functions and briefly describes their functionality.

Function Name	**Description**
`HsmIfAesSetKey`	Sets an entry of the HSM's AES key storage. Afterwards the key can be used for encryption/decryption
`HsmIfAesCryptECB`	Encrypts or decrypts an input block using a selected key in AES-ECB mode
`HsmIfAesCryptCBC`	Encrypts or decrypts the input using a selected key in AES-CBC mode
`HsmIfAesCryptCTR`	Encrypts or decrypts the input using a selected key in AES-CTR mode
`HsmIfAesCryptGCM`	Encrypts or decrypts the input using a selected key in AES-GCM mode

Table 5.1: HSM interface AES functions

For the CBC and CTR modes of operation an additional initialization vector (IV) needs to be provided as input to the function. The GCM mode offers authenticated encryption with associated data (AEAD) and requires an initialization vector. GCM encryption takes the plaintext and additional authenticated data (AAD) as input and computes the ciphertext and an authentication tag. Confidentiality and authenticity of the data are provided. The tag is a cryptographic checksum that is designed to reveal accidental errors and intentional modifications of the ciphertext and the AAD. The decryption process decrypts the ciphertext and checks the validity of the tag.

5.4.2.2 CMAC

Message authentication codes (MACs) are used to provide authenticity and integrity for a transmitted message or any other kind of data. Cipher-based message authentication codes (CMACs) are a keyed hash function based on a symmetric block cipher. An alternative message authentication code (MAC) achieving similar security goals [8, p. 2] is the keyed-hash message authentication code (HMAC). In contrast to the CMAC, the HMAC is based on a cryptographic hash function [1], e.g. SHA-256 [25]. The HSM has modules for SHA-256 generation and the AES block cipher. Both, CMAC and HMAC, would be possible to implement with hardware support. The decision to implement the CMAC algorithm is made based on the fact that the AES modules supports a read-only key storage.
The general CMAC algorithm is specified by NIST [38]. NIST does not specify a particular block cipher. IETF specifies the NIST CMAC algorithm with the AES-128 block cipher in RFC 4493 [8]. The selected algorithm is the one specified by IETF using the AES module of the HSM. RFC 4493 defines a message authentication code (MAC) generation and a MAC verification algorithm. Both are supported by the HSM interface. Tab. 5.2 lists the available C functions.

Function Name	**Description**
`HsmIfAesCmac`	Generates an AES-CMAC message authentication code for a provided message
`HsmIfAesCmacVerify`	Verifies an AES-CMAC for a provided message

Table 5.2: HSM interface CMAC functions

5.4.2.3 ECDH

The elliptic-curve Diffie–Hellman (ECDH) algorithm is specified in SEC 1 [15]. The algorithm is a key agreement scheme to establish a shared secret over a public channel. The shared secret is then used as cryptographic key. ECDH is based on the Diffie-Hellman key exchange method and elliptic curve cryptography (ECC). The supported elliptic curve parameters are specified in SEC 2: Recommended Elliptic Curve Domain Parameters [18]. The only available curve is the Standards for Efficient Cryptography Group (SECG) curve secp256r1 [18, Ch. 2.4.2] (also called NIST P-256 [28, p. D.1.2.3]).
Two primitives specified in SEC 1 are provided by the HSM interface: the key pair generation [15, Ch. 3.2.1] and the computation of the shared secret [15, Ch. 3.3.1] primitive. The generation of the key pair yields the secret key and the associated public key. The secret key is pseudorandomly derived from a non-deterministic seed. The public key is calculated by using the secret key and is basically a point on the elliptic curve. In a next step, the public key is published and the public key of the communication partner is retrieved. After the public key exchange, the shared secret is computed by both communication partners.
Tab. 5.3 lists the interface functions for ECDH operations. Further details about the algorithm implementation and the PRNG on HSM side are given in section 5.5.

Function Name	**Description**
`HsmIfEcdhGenPublic`	Generates an ECDH key pair
`HsmIfEcdhComputeShared`	Computes the shared secret

Table 5.3: HSM interface ECDH functions

5.4.2.4 ECDSA

The elliptic curve digital signature algorithm (ECDSA) is a digital signature algorithm specified by the American National Standards Institute (ANSI) [7] and based on elliptic curve cryptography (ECC). NIST specifies the ECDSA algorithm with NIST approved hash functions [28]. The digital signature standard (DSS) [28] uses a hash function to obtain a condensed version of the message to sign. The message digest is then used as input to the digital signature algorithm to generate the signature. The HSM interface offers functions for signature generation and verification (see Tab. 5.4). The pre-hashing of the message must be performed by the host application. As hash function the SHA-256 algorithm can be used. The only supported elliptic curve is the secp256r1 (NIST P-256) 256-bit curve requiring a 256-bit hash digest. Every signature generation requires a random number, the ephemeral private key. The generation of the ephemeral private key is done by the HSM internally seeding a pseudorandom number generator with the TRNG. After the signature generation the private key is discarded. Employing the same ephemeral private key for different signatures can lead to extraction of the long term signing private key [21]. Such a breach allows an adversary to generate digital signatures on its own.

Function Name	**Description**
`HsmIfEcdsaSign`	Computes the ECDSA signature of a previously-hashed message
`HsmIfEcdsaVerify`	Verifies the ECDSA signature of a previously-hashed message

Table 5.4: HSM interface ECDSA functions

5.4.2.5 SHA-256

NIST specifies the secure hash algorithm (SHA) with different versions and digest lengths [34]. The message digest is the output of the one-way hash function, a compressed representation of the message (input). The messages integrity can be determined. SHA-1 is not considered collision resistant [63, p. 18] and it is advised against a general use. The only considered cryptographically strong hash algorithm [63, p. 39] supported by the HSM hardware acceleration is the SHA-256 algorithm [34]. SHA-256 generates a 256-bit digest. Initial hash values and message preprocessing (i.e. padding and parsing into blocks) are defined by the standard and can optionally be performed by the HSM. Due to compatibility reasons regarding different host applications, the HSM interface offers two functions for hash generation (Tab. 5.5). The first function (`HsmIfSha256Process`) processes one input block only and preprocessing is delegated to the host application. The advantage is that the calculation is interruptible: a state variable is returned and as soon as a new message block appears, the generation can be continued. The second function (`HsmIfSha256`) processes an arbitrary message and returns the digest. In contrast to the first function preprocessing and setting the initial hash value is done by the HSM. The advantage of this function is that the calculation is entirely encapsulated in the HSM. A drawback is that the message must be complete before the hash generation can start.

Function Name	**Description**
`HsmIfSha256Process`	Processes one SHA-256 input block (512 bits), simply updates and returns the state and does not perform any padding
`HsmIfSha256`	Generates a SHA-256 hash digest of arbitrary data

Table 5.5: HSM interface SHA-256 functions

5.4.2.6 TRNG

The HSM interface offers a function to retrieve random numbers from the HSM's true random number generator (TRNG). The output of the physical random number generator of the HSM is directly returned to the host system. Tab. 5.6 lists the function to generate random numbers.

Function Name	**Description**
`HsmIfTrngGenerate`	Generates 32-bit non-deterministic (true) random numbers

Table 5.6: HSM interface TRNG functions

The random number obtained is compliant with functionality class PTG.2 [69]. Although PTG.2 generators can be used for certain applications where minor biases are negligible, in general it is recommended to use PTG.3 generators [63, p. 59].

5.5 HSM Application

An objective of this work is to develop a software focused on making use of the hardware building blocks of the Infineon Aurix Tricore's HSM. The available building blocks were introduced in section 4.3.1. The following chapter describes the architecture and functionalities of the developed application running on the HSM security CPU.
The HSM application offers functionalities to the host application. With the separated processor core and memory, a barrier against possible attacks on the host side is given. E.g. a buffer overflow attack on the host system can not corrupt the memory of the HSM secure CPU. Futhermore, keys and certificates can be stored in HSM memory to mitigate a leak of such information. The level of security is additionally increased by placing particular functionality (e.g. encryption and decryption process) in hardware [22, p. 18].

The implemented software architecture is analogous to a client-server model where the host system is the client and the HSM is the server. On request of the host system the HSM executes a function. The HSM offers services to the host system's application through the HSM interface (see section 5.4).
As outlined in section 5.4.1, the host application triggers an interrupt on the HSM as soon as a service is requested. The HSM application then computes the desired results. Via the memory interface buffer, addresses and parameters are exchanged. After an interrupt is triggered, the HSM application switches to the desired service. At the beginning of each service routine, the parameters set by the host system are checked. The HSM interface functions do not perform any parameter checks. E.g. null pointers as buffer addresses are forwarded to the HSM without signalling any error to the host application. Parameter checks are considered security relevant and are, therefore, performed in the secure CPU of the HSM. Passing e.g. null pointers to a function could cause the application to crash and manipulated values could be used to provoke a DoS. In case a parameter exceeds a boundary or is invalid, the HSM application returns an error code to the HSM interface using the memory interface. Performing such a parameter check at the HSM increases the level of security by encapsulating the execution in the Hardware Security Module (HSM) and by following the secure by design paradigm.

The secure by design approach intends to build in security in the system from the ground up, beginning with a robust architecture design [47]. It is not only necessary to design a robust security architecture which will ensure the system's software security, but it is also required to preserve the architecture during the evolution of the software [47]. Additionally, the correct implementation of the intended architecture is crucial in order to avoid introducing flaws. While both, architectural weaknesses on a higher level and software bugs on a code-level, could introduce security vulnerabilities, secure by design focuses on the first [47]. Since it is assumed that design weaknesses have a greater impact on the systems software security [47].

By accessing the memory interface, the HSM application reads and writes the RAM of the host system. The access is done directly without the optional caching. The caching provided by the HSM hardware increases the speed for some operations. Nevertheless, for this prototype implementation it is decided not to use the cache and to access the host memory directly. For future improvements the usage of the cache could be evaluated and tested. To access the host system's RAM, a memory window has to be set on HSM side. Each time a read or write operation is performed the window is priorly set.

A general paradigm of the HSM application is to avoid unnecessary memory copy operations. Input data provided by the host system is not copied to HSM RAM but directly to the HSM's hardware registers avoiding intermediate storage. This is possible whenever data is directly used as input for hardware accelerators without a need to pre-process. In cases where input data has to be preprocessed (e.g. padding), the smallest possible data block is copied to HSM RAM and then written to registers. The same approach is used for data generated by the HSM's hardware accelerators. Whenever possible, output data is directly moved from the hardware register to the host system memory without copying it to HSM memory first.

In the following the single functions provided by the HSM application are listed. The implemented algorithms are explained and, in case there are interface functions triggering the execution, they are enlisted. The functions are categorized in software modules provided by the HSM application. The software modules make use of the HSM's hardware modules outlined in section 4.3.1.

5.5.1 KMS

The key management system (KMS) module administers the keys stored on the HSM. The software module enables the storage of keys in the HSM's flash memory and in the AES hardware module. Keys stored in the AES hardware module are not readable but only writeable. The KMS module offers a function (table 5.7) to write AES-128 keys to the AES hardware module. The function is not directly accessible by the host system but is called by other functions of the HSM application (i.e. `AesSetKey`).

Function Name	**Description**
`KmsSetAesKey`	Writes a key to the AES-128 hardware module

Table 5.7: Functions of the KMS software module of the HSM application

5.5.2 Bignum

The Bignum module provides functions required to handle big numbers. Numbers used in the HSM application are up to 256 bits wide. The big numbers are represented as arrays of unsigned 32-bit integers. Tab. 5.8 lists the module's functions providing operations on such numbers. The module is used internally by the random number generators (TRNG and CTR_DRBG) to generate numbers within an interval. The CTR_DRBG module additionally uses the `BigNumInc` function to increase the counter value. The Bignum software module does not use any hardware accelerators and is not accessible by the host system.

Function Name	**Description**
`BigNumCmp`	Compare unsigned values
`BigNumInInterval`	Checks if unsigned value in an interval
`BigNumInc`	Increments an unsigned value by 1

Table 5.8: Functions of the Bignum software module of the HSM application

5.5.3 Hostmem

This software module provides functions to read and write the memory (RAM) of the host system. To enable data sharing between HSM and host, a shared memory is used. The following functions enable access of the HSM to RAM addresses of the host. The memory is accessed through a 64KB window, which is dynamically set. The functions write to the required register to set the memory window. The functions listed in table 5.9 are used by the other HSM application functions to exchange uncached data with the host system. The functions can be used to access HSM registers or variables. Different data units, reaching from single bytes to double words, are supported. It is also possible to exchange arrays of data between the two CPUs. For compatibility reasons it is necessary to support inversion of byte and word orders. Functions supporting hardware FIFOs repeatedly access the same register memory address but increase the host memory address. Memory registers are all 32 bits wide.

Function Name	**Description**
`hostwrite_byte`	Writes a single byte (8 bit) to a host memory address
`hostwrite_bytes`	Writes an array of bytes to a host memory address
`hostwrite_word`	Writes a single word (32 bits, 4 bytes) to a host memory address
`hostwrite_words`	Writes an array of words to a host memory address
`hostwrite_words_inverse`	Inverts the word order and the byte order of each word. Then writes the array of words to a host memory address
`hostwrite_fifo`	Writes the content of a FIFO buffer to a host memory address
`hostread_byte`	Read a single byte from a host memory address
`hostread_bytes`	Read an array of bytes from a host memory address
`hostread_word`	Read a single word (4 bytes) from a host memory address
`hostread_words`	Read an array of words from a host memory address
`hostread_words_inverse`	Read an array of words from a host memory address with inverted word order and byte order
`hostread_dword`	Read a double word (64 bit) from a host memory address

Table 5.9: Functions of the Hostmem software module of the HSM application

5.5.4 CTR_DRBG

This module implements a deterministic random bit generator (DRBG) according to NIST SP 800-90Ar1 [33]. The CTR_DRBG implements a pseudorandom number generator (PRNG) based on a block cipher [33, p. 48]. The block cipher is used in counter mode (CTR). Any NIST approved block cipher can be used. In this implementation, the AES block cipher with 128 bit key length is used because the algorithm is approved by NIST [56, p. 28] and supported by the HSM's symmetric crypto engine (see section 4.3.1.4). The implemented algorithm does not use a derivation function to derive a seed from the seed material, since with the HSM's TRNG a ”full-entropy input” is provided [33, p. 52]. NIST specifies interface functions to instantiate the generator, to generate random values and to reseed. Tab. 5.10 lists the implemented functions. The instantiate function initializes the internal state of the random number generator and takes an entropy value and a personalization string as input. The entropy values are bits from a randomness source. The HSM application uses the internal TRNG as randomness source. The personalization string is optional and, if given, combined with the entropy value to generate the final seed. The internal state is initialized depending on entropy input and personalization string. The internal state of the DRBG is the memory of the generator [56, p. 12], which is stored in the HSM's RAM. Since the output of the random number generator is derived from there, it is necessary to protect the internal state. The generate function yields a pseudorandom output. If additional input is provided, the internal state is updated before and after the random number generation. If no additional input is available, the internal state is updated after the random number generation. Other algorithms (i.e. ECDH, ECDSA) using the PRNG require random number within a certain range. Therefore, a function is provided which checks if the pseudorandom number is within an interval. The function interprets the generated values as unsigned integers and compares them with a given upper and lower boundary. If the generated number is too large or too small, the DRBG generate function is called again until the result is within the interval. It is important to emphasize that the behavior of this function is not deterministic. Due to the ”unpredictability” of the generated pseudorandom number the algorithm can get stuck within the loop. The chance of such a behavior increases, if the desired interval is small. Depending on the block cipher, the generator requires reseeding after a certain amount of generated random numbers. The reseeding function again requires an entropy value obtained from a randomness source and an optional additional input. The additional input, if provided, is combined with the entropy value and the internal state is updated.

Each time the internal state is updated or the output is generated, the block cipher is used. The counter is increased in software and the AES-128 hardware module is used in ECB mode. The use of a one-way function as state transition function and output function makes the CTR_DRBG a deterministic random number generator of the DRG.3 functionality class [24, p. 70, 90]. Using the HSM's TRNG, which is PTG.2 compliant [69], as entropy input results in a class PTG.3 random number generator [24, p. 79]. Class PTG.3 random number generators are ”appropriate for any cryptographic application ” [24, p. 79]. Especially when ECDSA signatures are generated, the quality of the random number used is crucial. Short term dependencies or biases of the PTG.2 physical random number generator are eliminated by the DRG.3-compliant postprocessing [24].

The CTR_DRBG software module is not accessible by the host system. No corresponding HSM interface function is implemented. The CTR_DRBG module is internally used by the HSM application to generate the ephemeral key for the ECDSA signature calculation and to generate the secret key for the ECDH key exchange.

Function Name	**Description**
`CtrDrbgInstantiate`	Instantiates the internal state of the DRBG with a seed
`CtrDrbgReseed`	Refresh the internal state with a new seed
`CtrDrbgGenerate`	Generates a pseudorandom value
`CtrDrbgGenInterval`	Generates a pseudorandom value within an interval
`CtrDrbgRunTest`	Runs a test of the DRBG with a test vector

Table 5.10: Functions of the CTR_DRBG software module of the HSM application

5.5.5 AES

The module implements the encryption and decryption using the Advanced Encryption Standard (AES) block cipher with 128 bit key length. Different modes of operation and a function to set the key are implemented. All provided modes of operation are also supported by HSM's AES hardware engine. The corresponding HSM interface functions described in section 5.4.2.1 are triggering the execution of the module's functions listed in Tab. 5.11. The encryption and decryption functions perform the following scheme. First the input parameters are checked. If the checks are successful, data from the host system is copied to the registers of the HSM's AES hardware engine. The hardware engine then computes the result and the HSM's CPU waits until completed. Subsequently, the values of the output registers are copied to the host system.

Function Name	**Description**
`AesSetKey`	Sets a key of the AES hardware module by calling `KmsSetAesKey`
`AesCryptECB`	Encrypts or decrypts an input block using a selected key in AES-ECB mode
`AesCryptCBC`	Encrypts or decrypts input data using a selected key in AES-CBC mode
`AesCryptCTR`	Encrypts or decrypts input data using a selected key in AES-CTR mode
`AesCryptGCM`	Encrypts or decrypts input data using a selected key in AES-GCM mode

Table 5.11: Functions of the AES software module of the HSM application

5.5.6 CMAC

The cipher-based message authentication code (CMAC) software module implements the AES-CMAC algorithm specified in RFC 4493 [8]. The module uses the HSM's AES-128 hardware engine to encrypt the blocks. The provided functions listed in Tab. 5.12 are triggered by the corresponding functions from section 5.4.2.2. The CMAC verification algorithm takes the message, the key and the authentication code (MAC) as input. The function then performs the same MAC generation as the `AesCMAC` function and compares the calculated authentication code with the provided one. If the MACs are equal, the provided authentication code is correct, otherwise the verification failed.

Function Name	**Description**
`AesCMAC`	Generate the MAC of a given message
`AesCMACVer`	Verify a given MAC of a given message

Table 5.12: Functions of the CMAC software module of the HSM application

5.5.7 ECDSA Curves

This software module stores the parameter of the supported elliptic curves. An elliptic curve E is specified by the following equation [18, Cap. 2.1]:

$$E : y^2 \equiv x^3 + ax + b \pmod{p} \tag{5.1}$$

Requiring the following sextuple of domain parameters T:

$$T = (p, a, b, G, n, h)$$

Where p is an integer specifying the finite field, a and b specify the curve over the finite field, $G = (x_G, y_G)$ is the base point, the prime n is the order of G and the integer h is the cofactor.

For this implementation the recommended verifiably random 256-bit elliptic curve domain parameters are chosen [18, Cap. 2.4.2]. The parameter set is denoted secp256r1 by the document and is equal to the parameter set NIST P-256 [28, p. D.1.2.3]. The curve uses the maximum supported bit length of the HSM's asymmetric cryptography engine (section 4.3.1.3). The curve parameters are imported by the ECDSA and ECDH software modules. No functions but global constants are provided by this module.

5.5.8 ECDH

The elliptic-curve Diffie–Hellman (ECDH) software module provides functions to generate a key pair and to compute a shared secret. The curve parameters from the ECDSA Curves (section 5.5.7) are imported. The key pair generation yields a private and a public key. The public key can be shared with the communication partner. The private key is generated using the CTR_DRBG module (section 5.5.4) seeded with entropy from the TRNG (section 5.5.11). With the private key, the public key is generated using elliptic curve cryptography with operations provided by the HSM's asymmetric cryptography engine (section 4.3.1.3). To compute a shared secret, the private key and the public key of the communication partner are used. Again, the HSM's asymmetric cryptography engine is used to calculate the shared secret value. The shared secret is established between two communication partners and can be further used as key material for symmetric encryption. Tab. 5.13 lists the functions provided by the module, which are triggered by the corresponding functions from section 5.4.2.3.

Function Name	Description
`EcdhGenPublic`	Generate a public-secret key pair
`EcdhComputeShared`	Compute a shared secret

Table 5.13: Functions of the ECDH software module of the HSM application

5.5.9 ECDSA

The elliptic curve digital signature algorithm (ECDSA) software module provides functions to generate and to verify a digital signature. The curve parameters from the ECDSA Curves (section 5.5.7) are imported. The functions require a 256-bit message digest as input. The signature generation and verification is done with operations provided by the HSM's asymmetric cryptography engine (section 4.3.1.3). The ephemeral private key for the signature calculation is randomly generated using the CTR_DRBG module (section 5.5.4) seeded with entropy from the TRNG (section 5.5.11). For the signature verification, the validity of a provided digital signature value is assessed. Tab. 5.14 lists the functions provided by the module which are triggered by the corresponding functions from section 5.4.2.4.

Function Name	Description
`EcdsaSig`	Generate a digital signature
`EcdsaVer`	Verify a digital signature

Table 5.14: Functions of the ECDSA software module of the HSM application

5.5.10 SHA-256

The SHA-256 software module uses the HSM's hash engine. The SHA-256 algorithm generates a 256 bit hash digest. Functions to process one input block and a message of arbitrary length are provided (see Tab. 5.15). An internal function, which yields the hash digest to the HSM, is implemented. This function can be used by other modules of the HSM application to process data with a the SHA-256 hash algorithm. Except from the internal functions, the functions listed in Tab. 5.15 are triggered by the corresponding functions described in section 5.4.2.5.

Function Name	Description
`Sha256Process`	Process one input block
`Sha256Internal`	Read input bytes from host and process them, return the hash digest to HSM, no parameter checks
`Sha256`	Read input bytes from host and write hash digest back to host

Table 5.15: Functions of the SHA-256 software module of the HSM application

5.5.11 TRNG

The TRNG software module provides functions to use the HSM's non-deterministic random number generator. The random numbers are used to seed the PRNG (section 5.5.4) and are made available to the host system. The host triggering the random number generation (using the interface function described in section 5.4.2.6) enables the HSM's non-deterministic random number generator. Each time a 32-bit random bit stream is completed, the interrupt service routine (see Tab. 5.16) is called. If the desired random numbers are returned to the host system, the generator is deactivated. For HSM-internal purposes, additional functions enabling the generator to be used in polling mode are provided. In polling mode, the generator is started and the HSM application waits until the result is ready. In case more than one 32-bit random number is requested the loop continues to read values from the generator's output register. Furthermore, an additional function which checks if the generated value is within an interval is implemented. Similar to the implementation outlined in section 5.5.4, the PRNG also runs in a loop until a valid random number is obtained. The functions provided by the module are listed in Tab. 5.16.

Function Name	**Description**
`TrngEnable`	Start random number generator
`TrngIRQHandler`	Interrupt service routine called after completed generation or when an error occurs
`TrngPoll`	Generate random numbers in polling mode
`TrngGenInterval`	Generate random numbers which are required to be within an interval

Table 5.16: Functions of the TRNG software module of the HSM application

6. Evaluation of Implemented Software

In this chapter the implemented concept is evaluated. The performance of the implemented algorithms is measured and compared to an open-source software-only implementation. Due to the limited time frame of the thesis, no penetration tests are performed.

6.1 Performance

Hardware implementations can not only improve the resistance against software attacks but also reduce the runtime. In this section different execution times of the HSM application are analysed and compared. The runtimes of the software implementation of this thesis and the runtimes of the mbed TLS [73] library for equal algorithms are compared. The mbed TLS library is executed on one core of the host systems Aurix Tricore CPU running at 300MHz. The cryptographic algorithms of the developed software solution are executed on the HSM on an ARM Cortex M3 with hardware accelerators. The HSM secure core runs at 100MHz.

The runtimes are measured with the System Timer (STM) of the Aurix Tricore microcontroller. The timer offers a 64-bit counter from which the lowest 32-bit are read. Before and after the call of the measured function, the timer value is read from the register and stored in a variable. The difference between the two time stamps is calculated and also stored in a variable. The granularity of the counter ticks is 10ns. A great deal of attention must be paid when using the lower 32-bits of the counter as an overflow occurs at some point and the difference value becomes useless. With the given granularity, the overflow happens at about 43 seconds ($2^{32} \cdot 10ns = 43s$) after the microcontroller's reset. This is acceptable as all measurements can be made within the time frame.

For the HSM software, the duration of the HSM interface function at the host system is measured. Consequently, the time needed to provide the results to the host application is measured. All the overhead regarding the interaction between HSM and host system is therefore included.

The runtime of each function is measured five times and the mean value is calculated. For functions intended to process data, the throughput is listed. By dividing the amount of test

data by the mean runtime value the throughput is calculated. The throughput is given in kibibyte ($1KiB = 1024$ *byte*) and mebibyte ($1MiB = 1024KiB = 1.048.576$ *byte*) according to IEC [16]. For an encrypting or decrypting function the throughput signifies how many plaintext or ciphertext can be processed within a second. For the random number generation, the throughput lists the amount of data which is generated in one second. For the SHA functions, the throughput signifies the amount of input data which can be processed.
Tab. 6.1 lists the results of the runtime measurements of the HSM interface functions. Tab. 6.2 lists the results of the mbed TLS functions. For functions to generate and verify a signature, to compute a shared secret, as well as for the SHA process, the frequency parameter is added. The frequency shows how many computations can be achieved in one second. That means e.g. that 219 signatures can be generated and 124 can be verified per second. The mbed TLS functions are called with the equivalent inputs and test data as the HSM interface functions.
The relative standard deviation (RSD) of the runtime measurements for most functions is below 1%. Such a low RSD value implies a sufficiently high confidence of the listed mean values (Tab. 6.1 and Tab. 6.2). Exceptions are the runtime measurements for `HsmIfAes-SetKey`, `HsmIfAesCryptECB` (ENC), and `HsmIfSha256Process`. The RSD values are between 3% and 10%. The confidence of the listed mean values for those functions is considered lower.

A comparison of the two implementations is made in Tab. 6.3. The runtimes of the corresponding functions are divided to obtain a value for comparison. As there is no result in Tab. 6.3 smaller or equal than one, the HSM interface is faster for all of the listed algorithms. Especially for functions using elliptic curve cryptography the HSM is faster. E.g. the computation of a shared secret is 64.64 times faster on the HSM than on the host core. For functions dealing with a small amount of data, such as AES-ECB or the processing of a single SHA-256 block, the runtime reduction is the lowest (between a factor of 1.29 and 2.4). It is plausible that for such small algorithms the overhead for the interaction between the two processors is large compared to the computation time of the result. For `HsmIfAesCmacVerify` and `HsmIfTrngGenerate` no corresponding function in mbed TLS is available.

Function Name	**Runtime (mean)**		**Throughput**	
HsmIf*	**Time** t_{HSM} (μs)	**Frequency** ($1/s$)	**Test data** (B)	MiB/s
`AesSetKey`	9.0	-	-	-
`AesCryptECB` (ENC)	10.6	-	16	1.438
`AesCryptECB` (DEC)	8.6	-	16	1.769
`AesCryptCBC` (ENC)	944.2	-	8192	8.274
`AesCryptCBC` (DEC)	1012.1	-	8192	7.719
`AesCryptCTR`	975.7	-	8192	8.007
`AesCryptGCM` (ENC)	1072.4	-	8192	7.285
`AesCryptGCM` (DEC)	989.3	-	8192	7.897
`AesCmac`	868.2	-	8192	8.998
`AesCmacVerify`	862.4	-	8192	9.059
`EcdhGenPublic`	4421.3	226.2	-	-
`EcdhComputeShared`	3877.0	257.9	-	-
`EcdsaSign`	4547.6	219.9	-	-
`EcdsaVerify`	8029.7	124.5	-	-
`Sha256Process`	16.8	59424.8	64	3.627
`Sha256`	414.3	-	8192	18.858
`TrngGenerate`	32150.1	-	2000	0.059

Table 6.1: HSM interface functions runtime measurement

Function Name	**Runtime (mean)**		**Throughput**	
mbedtls_*	**Time** t_{mbed} (μs)	**Frequency** ($1/s$)	**Test data** (B)	MiB/s
`aes_setkey_dec`	30.3	-	-	-
`aes_crypt_ecb` (ENC)	13.7	-	16	1.111
`aes_crypt_ecb` (DEC)	15.2	-	16	1.006
`aes_crypt_cbc` (ENC)	8392.2	-	8192	0.931
`aes_crypt_cbc` (DEC)	8487.2	-	8192	0.921
`aes_crypt_ctr`	8361.4	-	8192	0.934
`gcm_crypt_and_tag`	17764.8	-	8192	0.440
`gcm_auth_decrypt`	17762.8	-	8192	0.440
`cipher_cmac`	8577.8	-	8192	0.911
`ecp_gen_keypair`	254470.6	3.9	-	-
`ecdh_compute_shared`	250606.5	4.0	-	-
`ecdsa_sign`	101470.9	9.9	-	-
`ecdsa_verify`	361171.7	2.8	-	-
`sha256_update_ret`	40.3	24788.1	64	1.513
`sha256_ret`	4945.5	-	8192	1.580

Table 6.2: mbed TLS functions runtime measurement

HSM Interface Function	**mbed TLS Function**	t_{mbed}/t_{HSM}
`HsmIfAesSetKey`	`mbedtls_aes_setkey_dec`	3.36
`HsmIfAesCryptECB` (ENC)	`mbedtls_aes_crypt_ecb` (ENC)	1.29
`HsmIfAesCryptECB` (DEC)	`mbedtls_aes_crypt_ecb` (DEC)	1.76
`HsmIfAesCryptCBC` (ENC)	`mbedtls_aes_crypt_cbc` (ENC)	8.89
`HsmIfAesCryptCBC` (DEC)	`mbedtls_aes_crypt_cbc` (DEC)	8.39
`HsmIfAesCryptCTR`	`mbedtls_aes_crypt_ctr`	8.57
`HsmIfAesCryptGCM` (ENC)	`mbedtls_gcm_crypt_and_tag`	16.57
`HsmIfAesCryptGCM` (DEC)	`mbedtls_gcm_auth_decrypt`	17.96
`HsmIfAesCmac`	`mbedtls_cipher_cmac`	9.88
`HsmIfEcdhGenPublic`	`mbedtls_ecp_gen_keypair`	57.56
`HsmIfEcdhComputeShared`	`mbedtls_ecdh_compute_shared`	64.64
`HsmIfEcdsaSign`	`mbedtls_ecdsa_sign`	22.31
`HsmIfEcdsaVerify`	`mbedtls_ecdsa_verify`	44.98
`HsmIfSha256Process`	`mbedtls_sha256_update_ret`	2.40
`HsmIfSha256`	`mbedtls_sha256_ret`	11.94

Table 6.3: Comparison of runtime measurement

7. Summary and Reflections

In this chapter the results and possible future works are discussed.

7.1 Conclusions

As stated in the introduction, the main aim of this thesis was to apply cybersecurity-engineering techniques to an off-highway use case. Not only an analysis was conducted but solutions were implemented on an off-highway ECU. It has been shown that it is feasible to implement advanced secure communication protocols (TLS) on such embedded systems.
The thesis provides an introduction to the topic of off-highway security engineering and modern cryptography. Furthermore, the available theory is applied to a realistic example use case. A solution addressing the cybersecurity threats to such embedded systems is developed. The thesis has shown an application of the security pattern approach to an in-vehicle E/E system. A solution to enable secure diagnostic communication has been designed and developed. The developed embedded software provides a secure stack from the lowest layer up to the application layer where TLS is located. To remain within the time frame, instead of a diagnostic protocol, an HTTP server was integrated. The TLS protocol is integrated on an off-highway ECU making use of the microcontroller's (Aurix 2G) security coprocessor (HSM). Additionally, the software running on the security coprocessor was designed and developed. Several verification methodologies for cryptographic algorithms have been shown and a subset of freely available tools suitable for automotive penetration testing were presented. The focus was on maximizing performance and security. It has been shown that the use of the HSM and especially its hardware accelerators reduces the computation time for cryptographic algorithms. The runtime and throughput of the implemented software has been measured and listed in tables. Furthermore, a comparison with an open-source software-only implementation has been done. For every chosen algorithm, the developed implementation is faster than the software-only implementation. In particular the runtime for public-key cryptography is lowered significantly (up to 65 times faster). This enables new use cases especially in the field of real-time secure communications. Additional advantages of the dedicated security processor architecture are the better protected computations in hardware as well as the securely stored secrets (e.g. keys).

7.2 Future Work

It is clear that the software implementation of this thesis has to be viewed as a prototype. The reason being that the whole project was conducted entirely by a single person and not (peer-)reviewed by experts in the field. It is highly necessary to apply high standards, when security-critical software is developed. This includes code reviews and extended testing. Due to the limited time frame of a diploma thesis, the possibility to perform enhanced testing, validation and verification was not given.

The developed software supports a wide range of cryptographic primitives, but there is potential for extension. It has to be noted that the HSM application solely supports one elliptic curve. Therefore, a possible consideration would be to add more curve parameters, supported by the HSM asymmetric crypto engine. Furthermore, SafeCurves [68] and the German BSI [63] have security concerns about the implemented EC algorithms (ECDSA) and curve parameters (NIST P-256). On the other hand, the chosen algorithms are supported by TLS and the Infineon HSM. Hence, further research and evaluation is required.

The ECDSA algorithm supports the generation of a signature based on a hash digest. The hash function has to be called separately by the host application. The resulting hash digest needs to be forwarded to the signature generation/verification by the host application. This enables possible manipulation attacks on the hash result while it is stored in host memory. Implementing an additional function on the HSM, triggering internally the hash generation, would further increase security. The entire signature generation process from the input data to the signature value would be encapsulated in the HSM's secure CPU.

The functionality of the random number generator could be also extended. To give an example of what is meant: the PRNG is only available to the HSM application. In the future work the PRNG could be added to HSM interface and the host application developer could decide whether to seed the PRNG with the HSM's TRNG or with another random source.

Another limitation of the implementation is the blocking characteristic of the API. The host system waits in while loop until the HSM has completed the calculation. Due to the width of the communication register, the interface solely supports up to 32 commands. By extending the memory interface, it would be possible to increase the number of commands, allowing for more features.

When using public-key cryptography, a management of the public keys and certificates is required. A trust anchor, certificate authorities, certificate revocation lists, and certification paths might additionally be needed. The efforts related to key and certificate management should not be underestimated.

The HSM application does not support the storage of certificates. This functionality could be added in the future by extending the KMS software module of the HSM application. X.509 certificates [10] could be stored in the HSM's flash memory to further increase protection against manipulation.

Security aspects such as secure boot or secure firmware updates are not covered in this thesis but are considered crucial for the overall system security.

Security aspects regarding the transition from development to production and operation are not dealt with, but could be of further interest. Furthermore, additional security measures need to be applied considering for example debug ports and locking memory regions, to avoid tampering with the ECU's firmware.

Bibliography

[1] H. Krawczyk, R. Canetti, and M. Bellare, *HMAC: Keyed-Hashing for Message Authentication*, en, Library Catalog: tools.ietf.org, Feb. 1997. [Online]. Available: `https://tools.ietf.org/html/rfc2104` (visited on 06/15/2020).

[2] Simon Blake-Wilson, Hugh MacDonald, and Minghua Qu, *GEC 2: Test Vectors for SEC 1, Working Draft v0..3*, Sep. 1999.

[3] National Institute of Standards and Technology, „FIPS 197, Advanced Encryption Standard (AES)“, en, p. 51, Nov. 2001. [Online]. Available: `https://csrc.nist.gov/publications/detail/fips/197/final` (visited on 06/11/2020).

[4] ——, „Recommendation for Block Cipher Modes of Operation“, 2001. [Online]. Available: `https://csrc.nist.gov/publications/detail/sp/800-38a/final` (visited on 06/11/2020).

[5] H. Biel, „Diploma work of Christian Scheurer and Niklaus Schild“, en, p. 85, 2003.

[6] D. Whiting, N. Ferguson, and R. Housley, *Counter with CBC-MAC (CCM)*, en, Library Catalog: tools.ietf.org, Sep. 2003. [Online]. Available: `https://tools.ietf.org/html/rfc3610` (visited on 06/11/2020).

[7] American National Standards Institute, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, Nov. 2005.

[8] R. Poovendran and J. Lee, *The AES-CMAC Algorithm*, en, Library Catalog: tools.ietf.org, Jun. 2006. [Online]. Available: `https://tools.ietf.org/html/rfc4493` (visited on 06/15/2020).

[9] National Institute of Standards and Technology, „Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC“, Nov. 2007. [Online]. Available: `https://csrc.nist.gov/publications/detail/sp/800-38d/final` (visited on 06/11/2020).

[10] D. Cooper, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, en, Library Catalog: tools.ietf.org, May 2008. [Online]. Available: `https://tools.ietf.org/html/rfc5280` (visited on 06/21/2020).

[11] National Institute of Standards and Technology, „The Keyed-Hash Message Authentication Code (HMAC)“, en, National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST FIPS 198-1, Jul. 2008. DOI: `10.6028/NIST.FIPS.198-1`. [On-

line]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf (visited on 07/17/2020).

[12] E. Rescorla, *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*, en, Library Catalog: tools.ietf.org, Aug. 2008. [Online]. Available: https://tools.ietf.org/html/rfc5289 (visited on 07/19/2020).

[13] K. A. Scarfone, M. P. Souppaya, A. Cody, and A. D. Orebaugh, „Technical guide to information security testing and assessment.“, en, National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-115, 2008, Edition: 0. DOI: 10.6028/NIST.SP.800-115. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf (visited on 06/02/2020).

[14] The Institute of Electrical and Electronics Engineers, Inc., *IEEE Std 1619-2007: IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*, en. Place of publication not identified: IEEE, 2008, OCLC: 956669069, ISBN: 978-0-7381-5363-6 978-0-7381-5362-9. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=4493431 (visited on 06/11/2020).

[15] D. R. L. Brown, „SEC 1: Elliptic Curve Cryptography, Version 2.0“, en, p. 144, May 2009. [Online]. Available: http://www.secg.org/.

[16] DIN Deutsches Institut für Normung, *Quantities and units – Part 13: Information science and technology (IEC 80000-13:2008); German version EN 80000-13:2008*, de, Jan. 2009.

[17] ARM ltd., „Cortex-M3 Technical Reference Manual Revision r2p0“, en, p. 133, 2010.

[18] D. R. L. Brown, „SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0“, en, p. 37, Jan. 2010. [Online]. Available: http://www.secg.org/.

[19] J. Merkle and M. Lochter, *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*, en, Library Catalog: tools.ietf.org, Mar. 2010. [Online]. Available: https://tools.ietf.org/html/rfc5639 (visited on 06/12/2020).

[20] National Institute of Standards and Technology, „Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices“, 2010. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-38e/final (visited on 06/11/2020).

[21] *27C3: Console Hacking 2010*, Library Catalog: fahrplan.events.ccc.de, Feb. 2011. [Online]. Available: https://fahrplan.events.ccc.de/congress/2010/Fahrplan/events/4087.en.html (visited on 06/16/2020).

[22] Benjamin Weyl, Marco Wolf, Frank Zweers, Timo Gendrullis, Muhammad Sabir Idrees, Y. Roudier, Hendrik Schweppe, Hagen Platzdasch, Rachid El Khayari, Olaf Henniger, Dirk Scheuermann, Andreas Fuchs, Ludovic Apvrille, Gabriel Pedroza, Hervé Seudié, Jamshid Shokrollahi, and Anselm Keil, *Deliverable D3.2: Secure On-board Architecture Specification*, Aug. 2011. [Online]. Available: https://www.evita-project.org/Deliverables/EVITAD3.2.pdf (visited on 06/03/2020).

[23] European Commission – Information Society and Media, *EVITA E-Safety Vehicle Intrusion Protected Applications Fact Sheet*, Apr. 2011.

[24] W. Killmann and W. Schindler, „A proposal for Functionality classes for random number generators“, en, p. 133, 2011.

[25] M. Nystrom ¡magnus@rsasecurity.com¿, *Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512*, en, Library Catalog: tools.ietf.org, Mar. 2011. [Online]. Available: `https://tools.ietf.org/html/rfc4231` (visited on 06/15/2020).

[26] T. Schütze, „Automotive Security: Cryptography for Car2X Communication“, en, p. 16, Mar. 2011.

[27] M. Stamp, *Information security: principles and practice*, 2nd ed. Hoboken, NJ: Wiley, 2011, ISBN: 978-0-470-62639-9.

[28] Information Technology Laboratory, „Digital Signature Standard (DSS)“, en, National Institute of Standards and Technology, Tech. Rep. NIST FIPS 186-4, Jul. 2013. DOI: `10.6028/NIST.FIPS.186-4`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf` (visited on 06/09/2020).

[29] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*, en, 6th ed. Boston: Pearson, 2013, OCLC: ocn769141382, ISBN: 978-0-13-285620-1.

[30] P. Kleberger and T. Olovsson, „Securing Vehicle Diagnostics in Repair Shops“, in *Computer Safety, Reliability, and Security*, A. Bondavalli and F. Di Giandomenico, Eds., Cham: Springer International Publishing, 2014, pp. 93–108, ISBN: 978-3-319-10506-2.

[31] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, „Security Application of Failure Mode and Effect Analysis (FMEA)“, in *Computer Safety, Reliability, and Security*, A. Bondavalli and F. Di Giandomenico, Eds., Cham: Springer International Publishing, 2014, pp. 310–325, ISBN: 978-3-319-10506-2.

[32] W. Zimmermann and R. Schmidgall, *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur*, de. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, ISBN: 978-3-658-02418-5 978-3-658-02419-2. DOI: `10.1007/978-3-658-02419-2`. [Online]. Available: `http://link.springer.com/10.1007/978-3-658-02419-2` (visited on 06/01/2020).

[33] E. B. Barker and J. M. Kelsey, „Recommendation for Random Number Generation Using Deterministic Random Bit Generators“, en, National Institute of Standards and Technology, Tech. Rep. NIST SP 800-90Ar1, Jun. 2015. DOI: `10.6028/NIST.SP.800-90Ar1`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf` (visited on 05/30/2020).

[34] Q. H. Dang, „Secure Hash Standard“, en, National Institute of Standards and Technology, Tech. Rep. NIST FIPS 180-4, Jul. 2015. DOI: `10.6028/NIST.FIPS.180-4`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf` (visited on 06/12/2020).

[35] M. J. Dworkin, „SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions“, en, National Institute of Standards and Technology, Tech. Rep. NIST FIPS 202, Jul. 2015. DOI: `10.6028/NIST.FIPS.202`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf` (visited on 06/16/2020).

[36] D. C. Miller and C. Valasek, „Remote Exploitation of an Unaltered Passenger Vehicle“, en, p. 91, Aug. 2015. [Online]. Available: `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

[37] I. T. L. Computer Security Division, *Example Values - Cryptographic Standards and Guidelines — CSRC — CSRC*, EN-US, Library Catalog: csrc.nist.gov, Dec. 2016. [Online]. Available: `https://content.csrc.e1a.nist.gov/projects/cryptographic-standards-and-guidelines/example-values` (visited on 07/15/2020).

[38] M. J. Dworkin, „Recommendation for block cipher modes of operation: the CMAC mode for authentication“, en, National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST SP 800-38b, 2016, Edition: 0. DOI: `10.6028/NIST.SP.800-38b`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf` (visited on 06/15/2020).

[39] A. Langley and M. Hamburg, *Elliptic Curves for Security*, en, Library Catalog: tools.ietf.org, Jan. 2016. [Online]. Available: `https://tools.ietf.org/html/rfc7748` (visited on 06/12/2020).

[40] NXP Semiconductors, *SJA1105: 5-port automotive Ethernet switch, product data sheet Rev. 1*, Nov. 2016.

[41] C. Smith, *The Car Hacker's Handbook: A Guide for the Penetration Tester*, en. Warrendale, PA: SAE International, Mar. 2016, ISBN: 978-1-59327-703-1. DOI: `10.4271/1593277032`. [Online]. Available: `https://saemobilus.sae.org/content/B-981/` (visited on 07/14/2020).

[42] CCRA, *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Version 3.1 Revision 5*, Apr. 2017.

[43] Daan Keuper and Thijs Alkemade, *The Connected Car: Ways to get unauthorized access and potential implications*, 2017.

[44] ISO/IEC, *ISO/IEC 9594-8: Information Technology - Open Systems Interconnection - The Directory - Public-key and attribute certificate frameworks*, May 2017.

[45] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: a cyber-physical systems approach*, en, Second edition. Cambridge, Massachuetts: MIT Press, 2017, ISBN: 978-0-262-53381-2.

[46] NXP Semiconductors, *SJA1105P/Q/R/S: Objective short data sheet Rev. 1*, Nov. 2017.

[47] J. C. S. Santos, K. Tarrit, and M. Mirakhorli, „A Catalog of Security Architecture Weaknesses“, en, in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, Sweden: IEEE, Apr. 2017, pp. 220–223, ISBN: 978-1-5090-4793-2. DOI: `10.1109/ICSAW.2017.25`. [Online]. Available: `http://ieeexplore.ieee.org/document/7958491/` (visited on 06/18/2020).

[48] W. Stallings, *Cryptography and network security: principles and practice*, en, Seventh edition. Boston: Pearson, 2017, ISBN: 978-0-13-444428-4.

[49] Keen Security Lab, „Experimental Security Assessment of BMW Cars: A Summary Report", Tech. Rep., May 2018.

[50] Lukas Reier, *Automated Testing of Embedded Systems Software - Python Application Programming Interface for Test Case Designers*, Bachelor Thesis, Feb. 2018.

[51] C. Schmittner, G. Griessnig, and Z. Ma, „Status of the Development of ISO/SAE 21434", en, in *Systems, Software and Services Process Improvement*, X. Larrucea, I. Santamaria, R. V. O'Connor, and R. Messnarz, Eds., vol. 896, Series Title: Communications in Computer and Information Science, Cham: Springer International Publishing, 2018, pp. 504–513, ISBN: 978-3-319-97924-3 978-3-319-97925-0. DOI: `10.1007/978-3-319-97925-0_43`. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-97925-0_43` (visited on 04/09/2020).

[52] P. Castillejo, B. Curuklu, R. Fresco, G. Johansen, S. Bilbao-Arechabala, B. Martinez-Rodriguez, L. Pomante, J.-F. Martinez-Ortega, and M. Santic, „The AFarCloud ECSEL Project", en, in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, Kallithea, Greece: IEEE, Aug. 2019, pp. 414–419, ISBN: 978-1-72812-862-7. DOI: `10.1109/DSD.2019.00066`. [Online]. Available: `https://ieeexplore.ieee.org/document/8875170/` (visited on 07/17/2020).

[53] Infineon Technologies AG, *Automotive Cyber Security Compendium: Infineon Microcontroller AURIX*, 2019.

[54] Martin Brunner, M. Machold, and Bjoern Steurich, *Future requirements for automotive hardware security: Post EVITA Semiconductor Security Quo Vadis?*, 2019. [Online]. Available: `https://www.infineon.com/dgdl/Infineon-Future_requirements_for_automotive_hardware_security-Whitepaper-v01_00-EN.pdf?fileId=5546d4626df6ee62016e3709e03b03b1` (visited on 06/10/2020).

[55] C. Schmittner and G. Macher, „Automotive Cybersecurity Standards - Relation and Overview", en, in *Computer Safety, Reliability, and Security*, A. Romanovsky, E. Troubitsyna, I. Gashi, E. Schoitsch, and F. Bitsch, Eds., vol. 11699, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 153–165, ISBN: 978-3-030-26249-5 978-3-030-26250-1. DOI: `10.1007/978-3-030-26250-1_12`. [Online]. Available: `http://link.springer.com/10.1007/978-3-030-26250-1_12` (visited on 04/08/2020).

[56] E. Barker, „Recommendation for Key Management Part 1: General", en, National Institute of Standards and Technology, Tech. Rep. NIST SP 800-57pt1r5, May 2020. DOI: `10.6028/NIST.SP.800-57pt1r4`. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf` (visited on 06/21/2020).

[57] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, „Evaluation of CAN Bus Security Challenges“, en, *Sensors*, vol. 20, no. 8, p. 2364, Apr. 2020, ISSN: 1424-8220. DOI: 10.3390/s20082364. [Online]. Available: https://www.mdpi.com/1424-8220/20/8/2364 (visited on 05/25/2020).

[58] IANA, *Transport Layer Security (TLS) Parameters*, Jun. 2020. [Online]. Available: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml (visited on 07/19/2020).

[59] Infineon Technologies AG, *AURIX 32-bit microcontrollers for automotive and industrial applications: Highly integrated and performance optimized*, 2020. [Online]. Available: https://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/aurix-family-tc37xtp/ (visited on 06/01/2020).

[60] International Organization for Standardization, *ISO/SAE DIS 21434: Road vehicles - Cybersecurity engineering*, Feb. 2020.

[61] H. Martin, Z. Ma, C. Schmittner, B. Winkler, M. Krammer, D. Schneider, T. Amorim, G. Macher, and C. Kreiner, „Combined automotive safety and security pattern engineering approach“, en, *Reliability Engineering & System Safety*, vol. 198, p. 106 773, Jun. 2020, ISSN: 09518320. DOI: 10.1016/j.ress.2019.106773. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S095183201830499X (visited on 04/29/2020).

[62] Z. El-Rewini, K. Sadatsharan, D. F. Selvaraj, S. J. Plathottam, and P. Ranganathan, „Cybersecurity challenges in vehicular communications“, en, *Vehicular Communications*, vol. 23, p. 100 214, Jun. 2020, ISSN: 22142096. DOI: 10.1016/j.vehcom.2019.100214. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S221420961930261X (visited on 04/29/2020).

[63] B. für Sicherheit in der Informationstechnik, *Cryptographic Mechanisms: Recommendations and Key Lengths*, Jan. 2020.

[64] ——, *Cryptographic Mechanisms: Recommendations and Key Lengths, Part 2 – Use of Transport Layer Security (TLS)*, Jan. 2020.

[65] TTControl GmbH, *TTControl TTConnect 616 Datasheet V1.4*, 2020. [Online]. Available: https://www.ttcontrol.com/products/connectivity/advanced-platforms/ttconnect-616/ (visited on 05/30/2020).

[66] ——, *TTControl TTConnect 616 Flyer V3.1*, 2020. [Online]. Available: https://www.ttcontrol.com/products/connectivity/advanced-platforms/ttconnect-616/ (visited on 05/30/2020).

[67] United Nations Economic Commission for Europe, *Proposal for amendments to ECE/TRANS/WP.29/* Mar. 2020. [Online]. Available: http://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/GRVA-06-19r1e.pdf (visited on 06/08/2020).

[68] Daniel J. Bernstein and Tanja Lange, *SafeCurves: choosing safe curves for elliptic-curve cryptography*. [Online]. Available: `https://safecurves.cr.yp.to/` (visited on 07/16/2020).

[69] Infineon Technologies AG, *AURIX™ Security Solutions - Infineon Technologies*. [Online]. Available: `https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-security-solutions/` (visited on 08/30/2020).

[70] *Kali Linux Tools Listing*, en-US, Library Catalog: tools.kali.org. [Online]. Available: `https://tools.kali.org/tools-listing` (visited on 07/14/2020).

[71] *lwIP - A Lightweight TCP/IP stack - Summary [Savannah]*. [Online]. Available: `https://savannah.nongnu.org/projects/lwip/` (visited on 06/17/2020).

[72] *Readme file for the Controller Area Network Protocol Family (aka SocketCAN)*. [Online]. Available: `https://www.kernel.org/doc/Documentation/networking/can.txt` (visited on 07/14/2020).

[73] *SSL Library mbed TLS / PolarSSL*. [Online]. Available: `https://tls.mbed.org/` (visited on 06/13/2020).

[74] *Welcome to pyca/cryptography — Cryptography 3.0.dev1 documentation*. [Online]. Available: `https://cryptography.io/en/latest/` (visited on 07/15/2020).

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Datum	Unterschrift