iKoPA Integrierte Kommunikationsplattform für automatisierte Elektrofahrzeuge

Deliverable D5v2

Security Requirements and Architecture

Version number	2.0
Dissemination level	Public
Project Coordination	htw saar
Due date	31.12.2018
Date of preparation	21.12.2018

Funded by the

Federal Ministry
of Education
and Research

Project Coordination

Prof. Dr. Horst Wieker

Head of ITS Research Group (FGVT) at the
htw saar – Hochschule für Technik und Wirtschaft des Saarlandes,
University of Applied Sciences
Department of Telecommunications
Campus Alt-Saarbrücken
Goebenstr. 40
D-66117 Saarbrücken
Germany

Phone +49 681 5867 195
Fax +49 681 5867 122
E-mail wieker@htwsaar.de

Authors:

Richard Petri – Fraunhofer SIT

Table of Contents

List of Figures

1 INTRODUCTION

The interconnection of road traffic participants is a prerequisite for automated driving and brings many challenges for the safety and security of such a system. One of the goals of iKoPA is the combination of automated driving tasks with infrastructure-based data. While the complexity of modern communication standards is rising, the capabilities of potential attackers are matching up, as sophisticated tools for manipulating radio-signals are becoming affordable even for financially weak attackers. Work package 4.2 is a horizontal effort with the goal of developing a suitable security architecture for iKoPA in close cooperation with the other work packages.

The scope of this document is not that of “functional safety”, i.e. the protection against failing or misbehaving hardware or software components. The focus is the assessment of information security aspects and possible solutions, which protect against entities with malicious intent, such as external attackers or uncooperative iKoPA participants. To that end, the architecture developed in the context of iKoPA Work Package 1 was analyzed. General security requirements are already part of the iKoPA Deliverable D1 [1]. This document defines an attacker model and an according analysis of the defined use-cases, which investigates whether an attacker might subvert the given requirements. Appropriate measures for securing against attacks are introduced, including a Scalable IT Security Platform. This platform aims to be a suitable base software for the various types of hardware components used in iKoPA, providing strong integrity protection mechanisms and security services based on the second iteration of the Trusted Platform Module technology.

This document is structured as follows:

- **Chapter 2** describes the prerequisites technologies: The cryptographic methods used in the iKoPA architecture and their properties, as well as the TPM technology and the considered isolation methods for the IT security platform.
- **Chapter 3** covers the security analysis of the iKoPA use-cases and the architecture.
- **Chapter 4** introduces the concept architecture for the Scalable IT Security Platform.

2 PREREQUISITES

2.1 Cryptography

2.1.1 Hash functions

Hash functions [2] are commonly used to verify data integrity during message authentication. They serve, in simple terms, as a checksum, mapping arbitrary messages of finite length to a fixed length output called the hash-code, digest or simply the hash. In practice, a hash function is usually defined as $h: \{0,1\}^* \rightarrow \{0,1\}^n$, i.e. mapping bit strings of arbitrary finite length (e.g. the binary representation of a message) to a bit string of fixed length. Hash functions are not invertible, i.e. one-way functions, as multiple messages may have the same hash. As such, hash-functions must exhibit certain properties to be suitable for cryptographic applications (such functions are then called cryptographic hash-functions):

- Small changes in the input should result in large changes in the hash value, e.g. two slightly different inputs should have two completely uncorrelated hash values.
- It must be easy to compute the hash for any given input but computationally infeasible to find a matching input (called the *preimage*) for any given hash value.
- Additionally, it must be infeasible to find another input with the same hash value for any given input. Furthermore, it must be infeasible to find any two distinct inputs with the same hash value.

The National Institute of Standards and Technology (NIST) publishes the widely used set of functions, called the "Secure Hash Function" (SHA). The standards are continuously updated and the older SHA-1 standard is currently being phased out in many applications due to discovered flaws [3]. The more recent SHA-2 [4] or SHA-3 [5] standards are recommended at the time of writing.

2.1.2 Encryption

In the field of cryptography, encryption [2] is used to achieve confidentiality of information, for example, by encoding messages in a manner that only authorized parties are able to access it. In practice, an encryption scheme maps a *plaintext message* m (e.g. the binary representation of data in form of a bit string), to a ciphertext c (in most cases also a bit string). Decryption, the inverse process, then maps a ciphertext to a corresponding plaintext. Both encryption $E_e(m) = c$ and decryption function $D_d(c) = m$ are usually parameterized by a key, i.e. the encryption key e or the decryption key d respectively. A suitable encryption scheme should exhibit the following properties:

- It must be infeasible to find the corresponding plaintext of a ciphertext without the knowledge of the decryption key.
- Ideally, it should be impossible to find the corresponding key, even if plain- and ciphertext are known.

- The ciphertext should appear to be indistinguishable from random noise.

In iKoPA messages must be transferred from one party to another across communication links that can potentially be intercepted by unauthorized users. Accordingly, theses users might have an interest to read the content. This shall be protected by means of encryption. Two principally different methods are discussed and compared hereafter. Both methods are applied in the iKoPA project.

2.1.2.1 Symmetric Encryption

The term symmetric encryption describes schemes with encryption- and decryption functions, which use the same key (or where the decryption key can be easily derived from the encryption key and vice versa). Two commonly discernible categories of symmetric ciphers are block ciphers and stream ciphers. Block ciphers encrypt messages of some fixed length, e.g. 128-bit, and need to be employed in a *mode of operation*, which specifies a method as to how longer messages are to be partitioned into subsequent blocks and encrypted. A suitable block cipher for iKoPA is the AES cipher [6] in the CCM or GCM mode of operation. Stream ciphers, on the other hand, encrypt each digit of the message, e.g. each bit of a bit string, subsequently. A suitable example of a stream cipher is ChaCha20 as described in [7]. One drawback of symmetric encryption comes from the fact that also the receiver can produce the ciphertext. There is thus no proof of the origin of the ciphertext, see section 2.1.3.1.

2.1.2.2 Asymmetric Encryption

As opposed to symmetric encryption, asymmetric encryption relies on two separate keys for encryption and decryption [2]. One key can be made public, hence called the *public key*, and one is kept secret, named the *private key*. Due to this, these schemes are often called "public key encryption schemes". Messages can then be encrypted with the public key, and subsequently only be decrypted with the private key. Accordingly, it must be infeasible to derive the private key from the public key. Additionally, the sender of a message must ascertain the authenticity of the public key of the recipient. In practice, asymmetric schemes are only capable of encrypting short messages in the order of a few hundred bits, among others due to a considerable higher computational complexity compared to symmetric ciphers. Some applications, especially in the context of low-power embedded systems, require special hardware accelerators to achieve acceptable performance. Additionally, asymmetric schemes are often used in combination with symmetric schemes. In this case, a message is encrypted with a symmetric cipher and a random key. This random key is then encrypted with an asymmetric cipher and sent to the recipient.

Suitable examples are the RSA encryption scheme [8] or the "Integrated Encryption Scheme" (IES) and its elliptic curve based variant ECIES [2][1].

2.1.3 Authentication

The term authentication [2] requires an important distinction. It is often used in two different contexts: *identification* and *data origin authentication*. During identification, a proof of the identity of a communication party is established during an interaction between two parties. In contrast, the data origin authentication proves the origin of a message or document (e.g. the creator or sender). These schemes are further differentiated into *message authentication codes*, which are used in the context of symmetric encryption, and *digital signature* schemes, which are always asymmetric (public key based) schemes.

2.1.3.1 Digital Signatures

Signature schemes bind the identity of a party to a message (or document). This binding can then be verified by any third party to ascertain the origin and validity of that message. A digital signature scheme involves the following elements: a key-pair (e, d) generated by an entity, consisting of a public and a private key, a signing function S and a verification function V. The following requirements must hold:

- The signature function S maps the private key d and a message m to a signature s.
- The verification function V maps the public key e, a message m and a signature s to a Boolean value, i.e. it checks if a signature is valid.
- Modifying a message-signature pair (without affecting its verification outcome) must be impossible.
- No third party should be able to forge a signature, even after obtaining many valid message-signature pairs.
- Some applications may also require a signature to be “non-reputable”, i.e. the author of the signature should not be able to challenge the authorship.

Prominent example for signature schemes are the RSA signature scheme, and the DSA or its Elliptic Curve (EC) variant ECDSA. All of these schemes are suitable for the iKoPA project[2].

1 The RSA encryption scheme should only be used with a key length of at least 2048-bit, elliptic curve based schemes with at least 256-bit. These are suitable requirements at the time of writing and may need to be raised in the future.

2 The requirements for key lengths as detailed in section 2.1.2.2 also hold here.

2.1.3.2 Identification Schemes

An identification scheme is used *interactively* by two participants, and aims to prove the identity of one participant to the other [2].

- Two users A and B participate in an exchange, after which user B (the verifier) is convinced of the identity of user A (the claimant or prover).
- It is proven, that user A was active during such an exchange.
- Any third party must not be able to impersonate user A, even if such an exchange is intercepted.
- Depending on the application, it may be desirable that the verifier is not able to impersonate A after such an exchange, i.e. the identity is not transferable.

Any identification scheme uses one or more attributes of a claimant as identification. These attributes are usually classified in three categories: knowledge, possession and inherent properties. Knowledge could be represented by simple passwords, PINs or cryptographic keys. Knowledge is usually combined with something possessed, such as a secure smartcard or password generator. Inherent properties of a human user could be biometric attributes (e.g. a fingerprint). Computer chips can also offer inherent properties such as "physically uncloneable functions" (PUF), which are special circuits with properties subject to subtle manufacturing variations, and thus unique to every chip.

Note: In some applications related to iKoPA, it can also be possible that A and B are both, claimant and verifier. Then, mutual identification / authentication is needed.

2.1.3.2.1 Password based authentication

A weak but simple form of authentication are passwords. In its simplest incarnation, as shown in Figure 1, the claimant (A) authenticates itself to the verifier (B) by sending its identity[3] together with a password p. The verifier can then corroborate the correctness of the password by performing a lookup in a table of previously agreed upon passwords. This simple approach exhibits two problematic aspects:

- An attacker could impersonate the verifier to coerce the claimant to send the password, or the transmission could be intercepted.
- The password table stored at the verifier must be read and write-protected against unauthorized third parties.

[3] In many applications, something like an E-Mail address or username is used

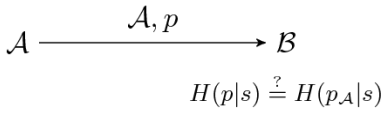

Figure 1: Authentication with passwords

To avoid eavesdropping or impersonation, the exchange must be performed in a private and authenticated manner, i.e. the claimant must first authenticate the verifier and establish an encrypted connection. To reduce the strong storage requirements for the password table, a one-way function – such as a hash function H – may be used to protect the password. Instead of storing the password itself, the verifier stores the hashed password, i.e. $H(p)$. During the password identification, the verifier now hashes the received password and compares it to the stored hash. To further protect against dictionary attacks, in which an attacker uses a dictionary of password and hash pairs to resolve a stored hash to a password, the verifier uses a "salt" value s during hashing. A salt value in this context is a random t-bit string, which is concatenated to the password prior hashing. This way, an attacker would require a dictionary with 2^t variations of each stored password. Further measures to increase the required workload of a dictionary attack could also include a fixed iteration count of H, i.e. by repeatedly using the hash function. Common key-derivation functions like "PBKDF2" or "scrypt" may be used to facilitate this. Password based authentication cannot fulfill all properties listed in Section 2.1.3.2. The verifier could for example impersonate the claimant after an exchange, as the verifier receives and stores the password. This is in most cases not problematic – for example if the verifier is a centralized service such as a website – and a well-accepted method of authentication. The security of this mechanism however relies on the privacy of the connection and prior authentication of B by A[4].

2.1.3.2.2 Challenge response authentication

A much stronger method of authentication is offered by "Challenge-Response" identification. During challenge-response protocols, the claimant proves knowledge of a secret without revealing it by introducing a time-variant challenge. The challenge is typically chosen by the verifier and sent to the claimant, who in turn constructs a response, which depends on the secret and challenge. One way to construct such a protocol is to use a symmetric encryption function with a previously shared key K, as shown in Figure 2. The claimant then encrypts the challenge and sends the result to the verifier, which decrypts and checks if the result matches the expected challenge. The encrypted challenge serves as a proof of knowledge of K and also proves that the claimant actively participated during the protocol. A stronger alternative to an encryption function with a shared key is the use

[4] Apart from the strength of the password, which could be enforced by one or both parties.

of a signature scheme. In this case, the claimant signs the challenge with a signature scheme and the verifier simply checks the signature.

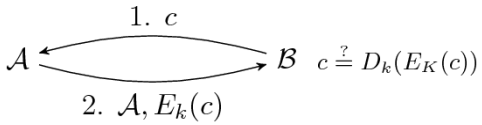

Figure 2: Basic challenge response authentication with a symmetric cipher

The security of this protocol relies on the security of the encryption function, and the freshness of the challenge. The challenge should ideally have an origin subsequent to the beginning of the protocol, i.e. is generated during the protocol and not known ahead of time. The challenge could for example be a strong random number of sufficient size. In this case, the chance of guessing the challenge prior the protocol must be negligible; otherwise, an attacker might guess the challenge and construct a valid response by interacting with the claimant. Another possible challenge could be a sequence number – which the verifier must check to be strictly monotonic – or a timestamp.

2.1.3.2.3 Distance bounding

Some applications, such as access control (e.g. door locks or car keys), have a further requirement. In these cases, the location of the claimant must be verified as well -- i.e. the distance between the claimant and the verifier. If this distance is not bounded, relay attacks (also called mafia attacks) are possible. The attacker could simply establish a relay between the claimant and verifier by forwarding challenge and response (see figure 3), and gain access. A prominent example is a radio attack on wireless electronic car keys [9]. The attackers simply amplified and relayed the radio signals between car and keys, and succeeded to unlock and start vehicles in the driveway, with the keys located further away (e.g. in the house).

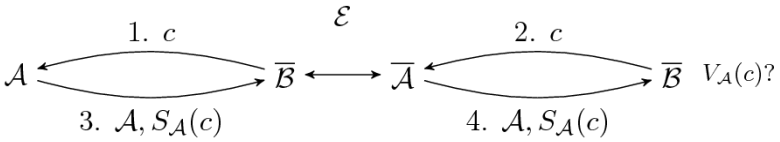

Figure 3: A relay attack on a challenge response authentication

Relay attacks can be mitigated, however with very limited effectiveness. A physical protection could be employed to hamper relay signals, for example a shielded card input in automated teller machines. A physical detection of a relay may be employed, e.g. by detecting amplified signals. Both of these measures depend on the application, but are likely to be circumventable or error prone. A last measure is a "presence detection" of the user, for example, if the car key requires a button press to perform the protocol instead of performing it automatically. Localization methods such as GPS could also be used, but may not be reliable in all situations and can be spoofed by an attacker [10].

Another method are distance-bounding protocols. These protocols usually consist of a repeated exchange of random bits. Both the verifier and claimant rapidly exchange random bits, while the verifier measures the round trip time of each message. All random

bits are then concatenated and serve as a challenge for a challenge response method. The round trip time then serves as a measure of distance for the verifier. As relay methods introduce a latency by nature, the verifier simply sets an upper bound for acceptable time, as the round trip together with the speed of light determines the maximum physical distance.

2.2 Trusted Computing

Perfect security and complete trust in a modern computing system is likely an impossible feat, as the complexity of systems is rising. To trust a running application, one must also trust what it is built upon, for example, an operating system. This in turn is possibly layered upon a bootloader, or a core firmware of the hardware (e.g. the BIOS in classical computers). Lastly, all this software is running on hardware, which is to be trusted. A complete audit of this chain, is likely an infeasible task. A common approach to tackle this problem is the reduction of what needs to be trusted for a system to remain secure to the minimum, the so-called "Trusted Computing Base" (TCB). One vital tool are Trusted Platform Modules (TPMs), which offer a "Root of Trust" in such systems.

2.2.1 Trusted Platform Modules

Hardware Security Modules (HSM) are an apt tool for enhancing the security of a range of applications. Typically, these devices provide means for the secure generation, storage and use of cryptographic keys and secrets, isolated from a potentially corrupted platform. Smartcards, for example, are widely deployed to enable secure identification of individuals. A TPM extends these use cases to a computing platform. As the identity of a modern computing platform depends on both its hardware and software, a smartcard is ill equipped to serve as means for identification of a computing platform. TPMs are intended to be permanently bound to the hardware platform, for example as a chip on a computers' motherboard. As such, a TPM is present during the boot time of the software and able to measure a "fingerprint" of the software running on the platform, which is stored in Platform Configuration Registers (PCR). This way, keys for encryption, authorization or identification can be bound to the platform in its entirety, i.e. its hardware *and* software.

The use of TPMs in iKoPA focuses on the TPM 2.0 standard [11], a complete overhaul of the TPM standard, with a range of powerful features (for a full outline see [12]):

- **Cryptographic Agility:** The previous TPM specifications were fixed to a very restrictive set of cryptographic algorithms, namely the SHA-1 hash function and RSA for encryption and signatures. A fixed set of algorithms was replaced by an agile approach. Instead of "hard coding" the algorithms, one chooses the desired algorithm out of an updateable list of algorithm identifiers, of which a TPM must implement at least a specific mandatory subset.
- **Enhanced Authorization:** TPMs allow the attachment of authorization policies to objects such as keys. Previously, this was restricted to passwords or PCR values.

The TPM 2.0 standard allows the creation of sophisticated and complex authorization policies (cf. section 2.2.2).

- **Non-Volatile Memory:** The capabilities of the non-volatile memory (NVRAM) were enhanced, enabling, among others, their use as monotonic counters or bitmaps.
- **Flexibility:** The implementation of a TPM is no longer restricted to a dedicated chip, but may also be implemented in software, for example, as a firmware TPM. Additionally, several profiles are defined, with varying feature sets.

A TPM can serve as a "Root of Trust" for several tasks, for example, a Root of Trust for Storage (RTS) or Reporting (RTR). For a TPM to be able to create a fingerprint of the software, it relies on each layer of software to cooperate. Each layer must make a measurement, i.e. calculate the hash, of the next layer and update the PCR values. For this approach to be effective, the bottom layer – called the "Core Root of Trust Measurement" (CRTM) – must be immutable. A simplified measurement chain of a system is shown in figure 4. Updating the PCR value "extends" the previous value. The old value of the PCR is concatenated with the hash of the next measurement, which is then hashed again and stored as the new PCR value. Thus, a PCR value serves as a fingerprint of the whole chain of software running on a platform, including its loading order. Because of this, the PCR extension is not commutative.

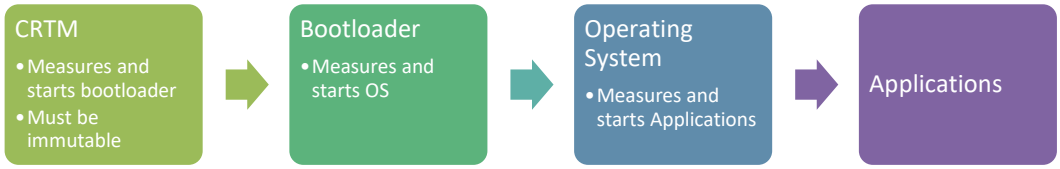

Figure 4: Simplified measurement chain of a computing system

2.2.2 Policy Enforcement

Enforcing policies is a classical use-case for an HSM, usually by enforcing a set of rules on cryptographic keys. An HSM may be used to generate and store keys with attributes, which the HSM will enforce, for example, whether or not a key can be migrated to another HSM. Keys are stored in secure memory on the HSM, which makes extraction by other means impossible or at least extremely costly. As mentioned in section 2.2.1, TPMs support a wide range of possibilities for enforcing policies on entities managed by it. In the context of TPM, this is called authorization, i.e. to access an entity in the TPM, such as a key or an object in NVRAM, one must first perform an authorization to do so. This feature was already present in the previous TPM 1.2 specification, but was improved for the TPM 2.0 specification under the term Enhanced Authorization. The standard defines 18 different policies, which can be logically combined (logical *and* as well as logical *or*) to create complex authorization policies.

A simple example of this is that of the full disk-encryption "BitLocker" as described in [12]. In this case, the TPM is used to create a key encryption key (KEK) with an attached policy, which specifies the PCR values to be present during decryption. This ensures, that the key

can only be used, if the operating system kernel or BIOS of the computer was not manipulated, as any manipulation would result in different PCR values. The KEK itself is used to de- and encrypt the full-disk encryption key. In [13], this idea is extended further with the enhanced capabilities of TPM 2.0: instead of just using PCR values, the TPM also keeps track of the version information of the running software via a monotonic counter in NVRAM. If an older software version is booted, the TPM will not unlock the decryption key, effectively preventing downgrade-attacks[5].

TPMs can also assist in enforcing policies in larger networks, for example, via remote attestation. During remote attestation, a TPM provides proof of the software configuration of the platform, by providing the digitally signed PCRs to a remote party. The protocol is similar to a signature based challenge response scheme (see section 2.1.3.2.2). The TPM first creates an Attestation Identity Key (AIK), which is essentially a key pair for a digital signature scheme. The remote party then creates a random challenge (also called nonce in this context) and sends it to the host of the TPM. The TPM can then perform a "quote" of the PCRs, i.e. it will sign the challenge together with the PCR values. This quote is then sent to the remote party, which can check if the PCR have the desired values.

2.2.3 Privacy CA

Every TPM contains an Endorsement Key (EK), which is a key pair for a digital signature scheme and is created during the manufacturing and initialization of the TPM. The manufacturer of the TPM usually provides a certificate for the public key of the EK, which can be verified by any third party. Any data signed by the EK can therefore be linked to a specific TPM, as the EK is unique to each TPM. To enable privacy sensitive use cases, the concept of Privacy Certificate Authorities (Privacy CA) was introduced. A Privacy CA adds a level of indirection to the certification of keys generated by TPMs. For example, in the case of remote attestation, the verifier may want to ascertain that the used AIK was generated by a compliant TPM. Signing the AIK with the EK would prove the validity of the AIK, however, this irrevocably links the AIK to a specific TPM, enabling the tracking of users. To protect the privacy of the users, the EK is instead used as a means of authentication to a Privacy CA. The Privacy CA uses the EK in a challenge response scheme to ascertain that an AIK was generated by a compliant TPM. If authentication is successful, the Privacy CA provides a certificate for the AIK, which any third party can verify. A user can then generate a new AIK before each use to conceal his identity.

[5] For a downgrade-attack, an attacker installs an older version of a firmware with known vulnerabilities on the hardware.

2.3 Application Isolation Techniques

Isolation is common technique to mitigate the impact of exploitable programming errors or malicious applications. There are varying degrees of isolation, but all aim to divide the resources of a computer into multiple execution environment. The origins are from "time-sharing" systems, with the goal to execute multiple applications without the possibility for one application to disturb another. For iKoPA, three different approaches were evaluated, which extend beyond the regular process separation of contemporary operating systems, detailed in the following sections.

2.3.1 Microkernels

Microkernels, such as the L4 microkernel family [14], follow the principle of reducing the TCB. In contrast to monolithic kernels, which incorporate all necessities of an operating system (e.g. drivers, file systems, memory management, etc.) inside the kernel, microkernels only implement the bare necessities. These kernels usually do not implement much more than process management, scheduling and an interprocess communication (IPC) mechanism. All other components of the system, such as the drivers and file systems, must be implemented as processes. This usually comes at the price of lower performance, as much more IPC is necessary than in systems with monolithic kernels. The potential gain lies in a potentially more secure implementation. As monolithic kernels include many more components, any programming error inside these may potentially be used by a malicious application to subvert the protections (e.g. process or memory separation) the kernel provides. As the code base of a microkernel is considerably smaller and as such, the likelihood of programming errors that might subvert the process separation is smaller. Furthermore, the "seL4" microkernel [15] was even formally verified for correctness.

2.3.2 Virtualiziation

Another technique that can be considered for separating applications is virtualization [16], whereby multiple "virtual machines" are run on top a single physical machine. In this case, the physical machine runs a "hypervisor", a program that manages the virtual machines, maintaining a strong separation of the resources (e.g. the memory regions) of each virtual machine. Depending on the degree of virtualization, a hypervisor can be capable of running multiple instances (called guest systems) of full operating systems. Usually, this is used to achieve higher utilization of a computing system, by consolidating multiple services on a single physical machine, while maintaining a separation. This may, however, also be used to achieve a strong separation between applications on an embedded system in the context of iKoPA. The drawback of virtualization is its overhead and the need for hardware support. Dedicated hardware support for virtualization is available from most

hardware manufacturers, for example, “Intel VT”[6] or “ARM Virtualization Extensions”[7]. However, while these hardware extensions lower the computational overhead for virtualization, multiple guest operating systems still incur a significant storage overhead.

2.3.3 OS-Level Virtualization

Some operating systems also allow a lighter form of virtualization, called operating-system-level virtualization [17]. Prominent examples are “LXC”, which is built on top of the Linux Kernel Namespacing functionality, or the “Jail” [18] functionality provided by the FreeBSD operating system. Instead of running a full operating system inside a virtual machine, a number of distinct user space instances, often referred to as containers, are isolated by the operating system kernel. The main advantage over a full virtualization approach is that of resource efficiency, as all containers share a single operating system kernel. Furthermore, no additional hardware support is required to achieve acceptable performance. The drawback is a loss of flexibility, as this approach does not allow the virtualization of different operating systems, i.e. a Linux based environment is only capable of running other Linux environments inside containers.

[6] http://www.intel.de/content/www/de/de/virtualization/virtualization-technology/intel-virtualization-technology.html

[7] https://www.arm.com/products/processors/technologies/virtualization-extensions.php

3 SECURITY ANALYSIS OF IKOPA USE-CASES

A common problem among the defined use-cases of iKoPA[8] is authentication in its two different forms: identification, e.g. if a proof of identity of an actor must be established, and data origin authentication, e.g. if the origin of data must be verified. The impact of an attack varies between the use-cases and thus the required level of protection and assurance must be appropriate. Furthermore, in cases where identification is necessary, the privacy of the user must be protected. In the following sections, an attacker model is defined and used to guide an analysis of the following scenarios, which are part of iKoPA:

- Acquiring information about available parking (part of use-case 1)
- Receiving and using traffic light forecast information (part of use-case 3)
- Automated parking within car park using camera based location tracking (use-case 5 and 7)
- Reservation of parking and/or charging of an electric vehicle (part of use-case 1)
- Receiving entry clearance for car park (part of use-case 4)
- Get access to parking lot via RFID Identification (part of use-case 8)
- Receiving state of charge information (part of use-case 10)
- Requesting a parked vehicle (part of use-case 11)

For each of these scenarios, an analysis on possible attack targets is discussed and – if applicable – a suitable protection mechanism is proposed.

3.1 Attacker Model and Risk Assessment

The first step to finding and analyzing possible risks and weaknesses is the definition of an attacker model. In this section, the resources, abilities and possible motivations of an attacker are discussed. In the cases described above, the points that might enable an attack (the attack vectors) are the following: the communication channels, and possibly any hardware that might be accessible to an attacker. Therefore, it is assumed, that an attacker is capable of the following:

- Observing all communication over most wireless communication channels
- Manipulation or injection of messages on most wireless communication channels
- Manipulation of firmware running on publicly accessible hardware components

The first two aspects have become of increasing relevance in recent years, with tools like Software Defined Radios (SDR) becoming cheaper and more accessible. Models such as

[8] For a detailed description of the use-cases of iKoPA, see Deliverable D1 [12].

the "HackRF One"[9] or "LimeSDR"[10] are priced below the four-figure mark, attainable even for financially weak attackers. These tools allow the observation and interpretation of a very wide range of radio signals, as well as sending of arbitrary signals. It is assumed that an attacker is able to intercept and inject (or repeat) V2X messages sent between vehicles or roadside stations. All other wireless communication channels are affected by this as well, such as WiFi, RFID, DAB+ or cellular communication. The latter capability of the attacker, assumes that an attacker might, for example, manipulate the firmware running on a hardware component installed in a publicly accessible place. This could include hardware components such as traffic lights, barriers, chargers, or cameras. Outside of the scope of the attacker model is the following:

- Breaking contemporary cryptographic schemes
- Extracting secrets from secure storage, such as RFID tags or TPMs
- Attacking system components such as services running in data centers

Two separate types of attackers guide the analysis of the possible motivations (i.e. the attack target): an external and an internal attacker. An external attacker acts from outside of the iKoPA system, for example, with the motivation of impersonating valid users for personal gain. An internal attacker on the other hand, may be a misbehaving party of the iKoPA system. Other possible goals of either type of attacker might be, for example, tracking the users of the system, corporate espionage or manipulating users for personal gain.

Figure 5: Example of an attack tree used for risk assessment

The risk analysis itself is structured as a tree, as shown in figure 5. At the root of the tree is the goal of an attacker. Each subtree is a path that an attacker might take to achieve that goal. The nodes are color-coded to signal problems, which will be addressed in the description. Green colored nodes are attack vectors, which are either easily secured or inherently secure due to the used technology. Yellow colored nodes are problems that can be addressed with further measures. Red colored nodes are either technologically unsolved, or not within the scope of the iKoPA project.

[9] https://greatscottgadgets.com/hackrf/

[10] https://myriadrf.org/projects/limesdr/

3.2 Analysis of Use Cases

3.2.1 Acquiring information about available parking

Before a parking can be reserved, information about available opportunities must be gathered and presented to the user as possible choices, which is part of use-case 1.3. For the purpose of a security assessment, an attacker with the intent of personal gain is assumed. For example, a malicious party of the iKoPA system may cause other car parks to appear full, to attract more customers. Figure 6 shows the attack tree for such a scenario. The root goal of the attacker would be to modify the TPEG information received by users. This might either be possible at the level of data acquisition, or after the transmission. Attacking the acquisition is not possible under the assumption of the attacker model defined above, as this would involve attacking a data center of either the car park or the TPEG playout. Modifying the transmission, however, is possible under the defined attacker model.

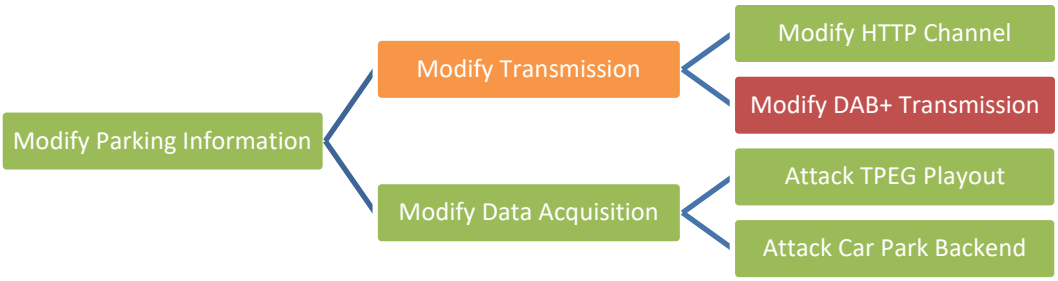

Figure 6: Attack tree for parking information acquisition

In [19] it is shown, that DAB radio signals can be created or manipulated with SDR tools. The work focused on the "fuzzing" of DAB+ radio receivers, which would deliberately introduce errors in the signal to induce an invalid state in the processing of the signal, possibly allowing an attacker to gain control over the receiver. It is further conceivable that an attacker might send valid (albeit forged) data to the receiver to spread false information for personal gain. At the time of writing, the TPEG standard offered no protection against such an attack. While the DAB+ and TPEG protocol itself involve strong signal integrity measures for error correction, no data origin authentication is performed. In the future, authentication measures like digital signatures should be introduced to the protocols. In the meantime, it is advisable not to perform any safety relevant action based upon data received via DAB+. This is not the case in the context of iKoPA, as the problem is further alleviated by using a hybrid approach for the dissemination of data. The information is also spread via internet and HTTP protocol, which can be easily protected via conventional protection mechanisms like TLS. Information gathered via this method must therefore be favored to that received via DAB+, which would override any maliciously introduced discrepancies.

3.2.2 Receiving and using traffic light forecast information

This use-case is in most aspects very similar to section 3.2.1, as it also involves gathering information, which is processed by a central service and then disseminated to end users. The type of information, however, is safety critical, as it is used to create speed advisories to drivers. In the worst case, a goal for an attacker might be to manipulate the traffic light forecast (TLF) information to cause an accident. The attack tree for this case is shown in figure 7. Modifying the data transmission in this case is not likely to be possible. A central transmission via internet can be protected with conventional means like TLS. An attacker might attempt to modify the V2X (ETSI ITS G5) transmission, e.g. by sending malicious V2X messages, which could be possible with a SDR. The V2X protocol, however, offers protection via digital signatures, which an attacker cannot forge. Retransmission of old messages may be used to manipulate the TLF information visible to the vehicle, but the impact of this vector is alleviated by the time stamps within the messages and ignoring messages beyond a certain age.

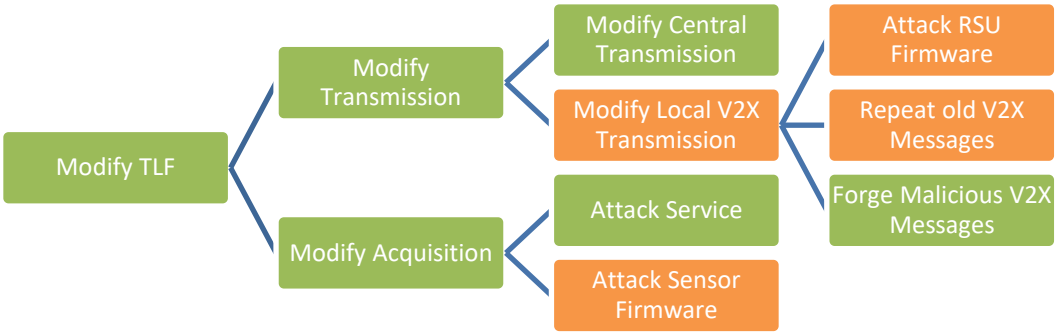

Figure 7: Attack tree for TLF information

Another attack vector is that of an attack on the firmware of accessible hardware, such as the sensors involved in the process or the RSU transmitting the V2X messages. A malicious firmware may enable the modification of TLF data (or sensor data from which TLF data is derived) before any transport security mechanism. These attack vectors can be thwarted by using a firmware integrity mechanism. A "secure boot" approach, for example, would prevent any modified firmware from booting. One step further would be securing the firmware with TPM technology, which would allow the remote attestation of the firmware status. This approach is preferable in the case of sensors.

3.2.3 Automated parking within car park using camera based location tracking

Automated driving involves even more safety critical information than a TLF. Use-case 7 describes a camera based positioning system inside a car park, which tracks the positions of vehicles inside a car park. The positioning information is then transmitted to the vehicle performing an automated parking maneuver. An attack on this system may therefore impair the safety and cause property or even personal damage. The main goal of an attacker might be to cause an accident, the attack tree for such a scenario is shown in Figure 8.

This involves similar problems as described in section 3.2.2. A firmware integrity protection mechanism must be applied to the sensors of the system, in this case the cameras, as well as to the RSU transmitting the position and the components inside the vehicle. The calculation of the position should be performed on components, which are inaccessible to attackers (e.g. a data center).

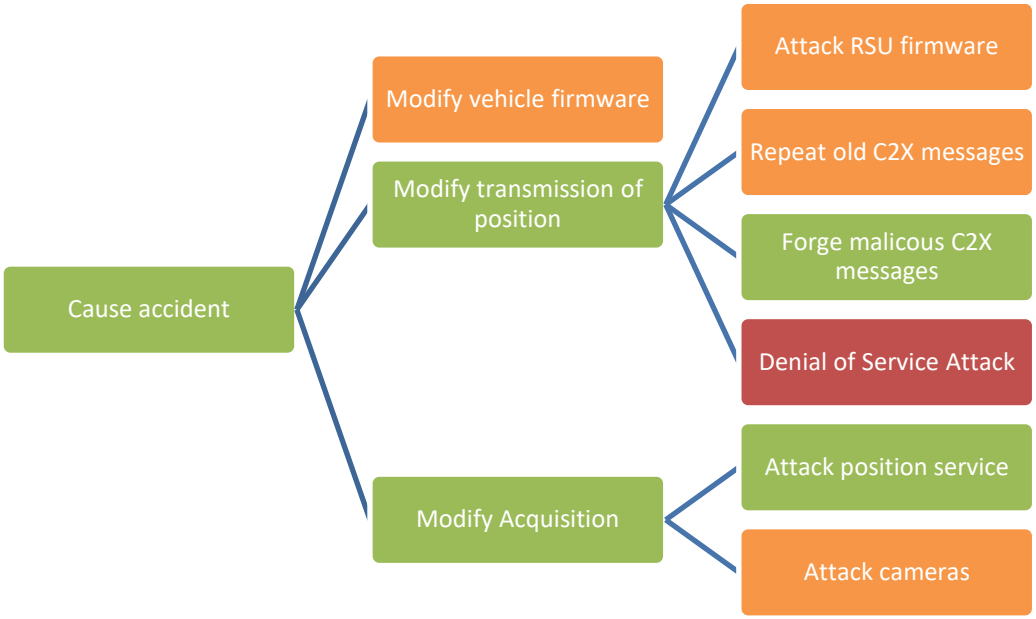

Figure 8: Attack tree for the automated driving process

The calculated position of the vehicle is then transmitted via V2X messages to the vehicle itself. As previously described, the forgery of malicious V2X messages is thwarted by the digital signature scheme offered by the protocol. Repetition of old messages by an attacker, however, is much more problematic than in the case of a TLF. An attacker might reuse old messages to confuse the position, or use messages intended for another vehicle. A replay of old messages may be thwarted by including a sequence number inside messages. A vehicle is then able to detect older messages by checking the sequence number. Lastly, an attacker might perform a denial of service attack by blocking or disrupting all communication at an opportune time. Such an attack cannot be defended against, and appropriate safety precautions should be taken. For example, the vehicle might stop as soon as position messages cease.

3.2.4 Reservation of parking

Apart from acquiring information about parking (see section 3.2.1), another aspect of use-case 1 is the reservation of parking. Here, two different attacker goals are considered: the impersonation of another user by an external attacker and the tracking of users over multiple reservations. In the following section, a mechanism is described, which aims to prevent such attacks, and is analyzed in the subsequent section.

3.2.4.1 Privacy protection of users

One of the main goals is the protection of the privacy of users of the iKoPA system. We therefore propose an architecture with a trusted third party, which masks the identity of users: a "registration service". The role of such a service is similar to that of a "Privacy CA" in the context of the TPM technology (see section 2.2). The mechanism works as follows: the registration service is the sole party of the system, which has the means to directly identify and authenticate users, for example, with a common password authentication scheme. The service will then allow users to create a certified pseudonym. As other parties of the system implicitly trust the registration service, they will accept any pseudonym certified by it. The process for creating a pseudonym is shown in figure 9. It is assumed, that each user of the iKoPA system is registered with a unique username and a chosen password, which can be used for authentication by the registration service. The smartphone of the user then creates a keypair for a digital signature scheme with public key e_U and private key d_U. The public key is then sent to the registration service together with the username and password for authentication. The registration will then check the username and password to authenticate the request and certify the public key. The certificate $\overline{e_U}$ is sent back to the smartphone.

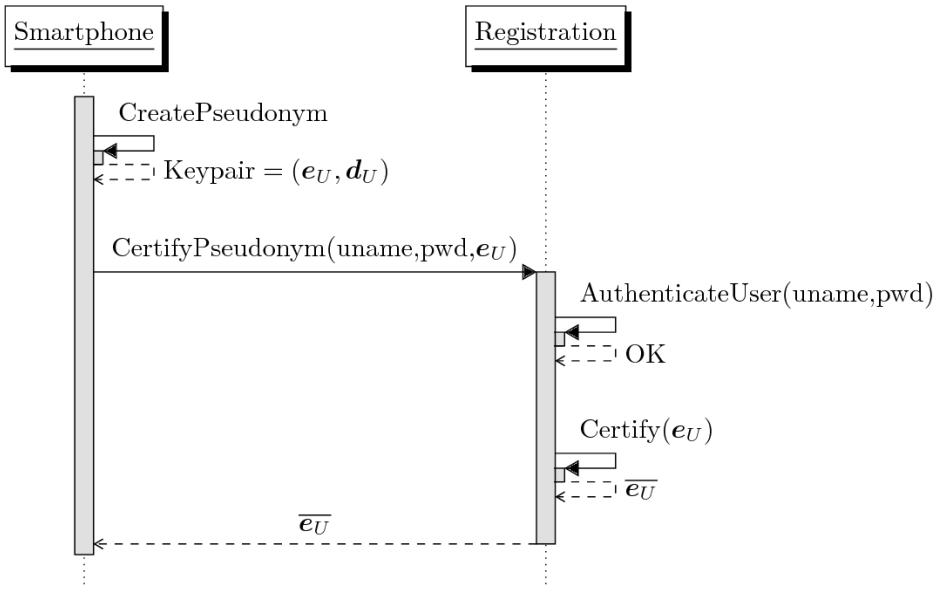

Figure 9: Creation of a pseudonym

On a technical level, such a registration service can be implemented with a common PKI infrastructure, which establishes trust for user generated signature key pairs, for example, based on X.509 certificates [20]. The smartphone will create a keypair for signing and a certificate signing request (CSR) [21]. The registration server will create a X.509 certificate for the public key. This certificate is then used as a means for authentication of the user, for example, enabling a mutually authenticated TLS connection or signing data. Any other party can check the certificate to establish trust and assume that it interacts with an authenticated user. In the context of the use-case description, the certificate serves as the

"voucher" presented to the reservation service. The certificate acts as a pseudonym for the user, and is called the pseudonym certificate or pseudonym in short, in the following.

3.2.4.2 Reservation mechanism

The reservation process itself is based upon the pseudonym functionality described above. The user is authenticated with the pseudonym certificate, either by requiring reservation requests to be signed with the private key belonging to the certificate, or by using the certificate to create an authenticated channel to the reservation service. To thwart tracking of users, a pseudonym should only be used once per reservation, with a new pseudonym created for each instance. A goal of an attacker would be to impersonate a user for personal gain. The risk assessment for this is shown in Figure 10. The highest risk emanates from the user and the smartphone itself. If an attacker seizes control of the smartphone (either physically, or by modifying the firmware), the username and password can be captured and used. The risk varies with the type of access an attacker has gained. Such an attack can be mitigated by secure storage mechanisms provided by most smartphone operating systems [22], but not totally excluded. This is, however, outside of the scope of iKoPA. Another possibility is "social engineering", i.e. coercing a user to reveal the credentials. A prominent example of social engineering is "phishing", in which an attacker often disguises as a trustworthy entity, for example by sending a spoofed e-mail [23].

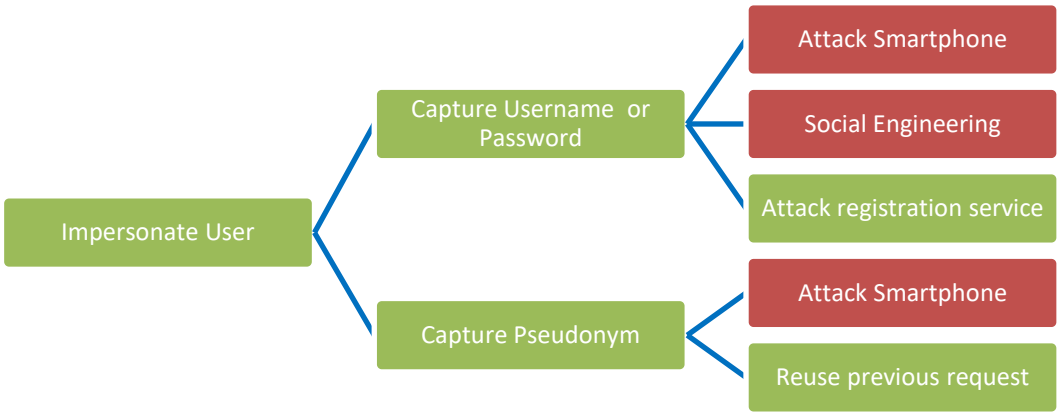

Figure 10: Attack tree for user impersonation during reservation

3.2.5 Receiving entry clearance for car park

The privacy protecting identification mechanism described in section 3.2.4 may also be used in use-case 4. The process is shown in figure 11. The pseudonym certificate is attached to the reservation request and must be sent to the barrier (or the RSU of the barrier). As the vehicle approaches the barrier, it will send an authentication request to the barrier together with the pseudonym certificate. The barrier must check the validity of the pseudonym certificate and whether it is attached to a reservation request. The barrier will then perform a challenge response authentication by sending a challenge to the ve-

hicle, which must be signed with the private key corresponding to the pseudonym certificate. As the private key resides on the smartphone, the challenge must be forwarded accordingly. After the barrier checked the response, the authentication sequence is complete and actions can be triggered accordingly.

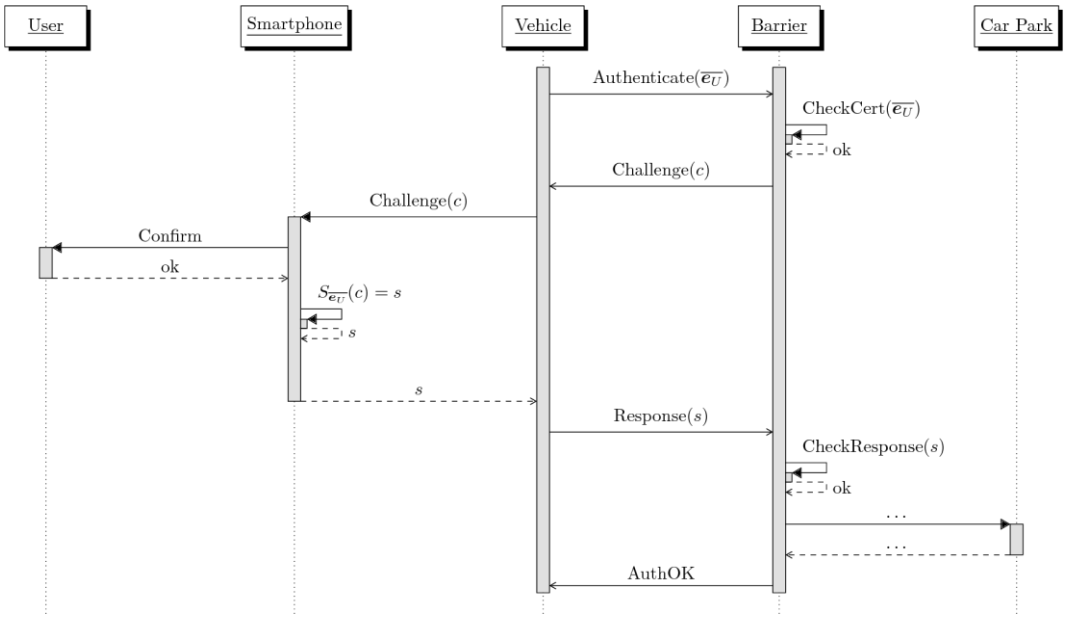

Figure 11: Authentication sequence for receiving entry clearance for car park

The risk assessment of this procedure is shown in figure 12. The most problematic risk is again that of an attack on the smartphone, which is outside of the scope of iKoPA. Aside this, forging any message is protected appropriately, as V2X messages are all signed and optionally encrypted. Relaying the communication between the vehicle and the barrier, however, is a possible attack vector. An attacker may even relay the communication over larger distances. As such, the sequence includes an additional safeguard: a confirmation step of the user. The smartphone must not perform the authentication step automatically, but notify the user that an authentication is possible. Only upon confirmation by the user, the authentication is completed. The user should only confirm the authentication, when in front of the barrier. Employing firmware integrity measures is also advisable, as attacking the firmware of the car park RSU is also a possible attack vector.

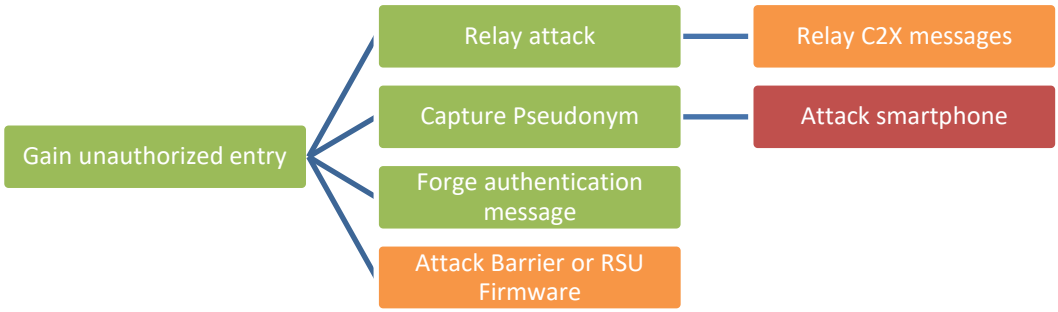

Figure 12: Attack tree for unauthorized entry

3.2.6 Get access to parking lot via RFID Identification

The identification method provided by RFID has a considerable disadvantage in comparison with the procedure detailed in section 3.2.5: the RFID tags are only capable of performing challenge response authentication schemes with symmetric cryptographic ciphers. Hence, the authentication is performed with a pair of secret keys, also causing the identity to be transferable to anyone who gains knowledge of the key. To that end, the key must only be used by the most essential components to the process.

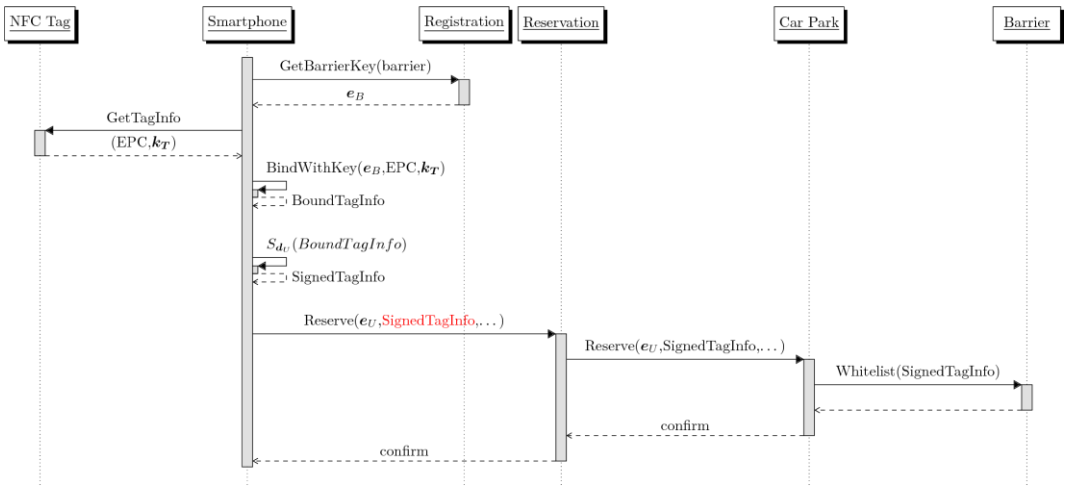

Figure 13: Extended reservation sequence for RFID authentication

To facilitate the authentication, the secret key k_T used by the RFID tag must be transported to the RFID reader of the barrier (or charger), for example, during the reservation process. Figure 13 shows an extended sequence for the reservation, with a secured transport of the key to the barrier. The user receives both a RFID and a NFC tag, both of which contain a copy of k_T. The NFC tag contains the key in a format readable by the smartphone of the user. The concept then uses asymmetric encryption to "bind" the RFID key for the barrier. The reservation service is used as a trusted third party, which holds the public key e_R of the barrier for encryption. The smartphone then encrypts k_T and other identifying information about the RFID tag (such as the EPC) with the barriers key, and attaches it to the reservation, which is forwarded to the barrier. As only the barrier owns the corresponding decryption key d_R for the public key e_R, only it will be able to decrypt the key and information about the RFID tag and use it.

The authentication process is shown in figure 14. To enhance the privacy protection, the RFID Tags considered for iKoPA use a two-stage authentication. A RFID reader must first perform an authentication with a group-key k_G. After this, the RFID Tag will unlock identifying information, such as the EPC. The reader will then decrypt the corresponding secret key k_T to perform the final authentication of the tag, and open the barrier (or unlock the charger) if successful.

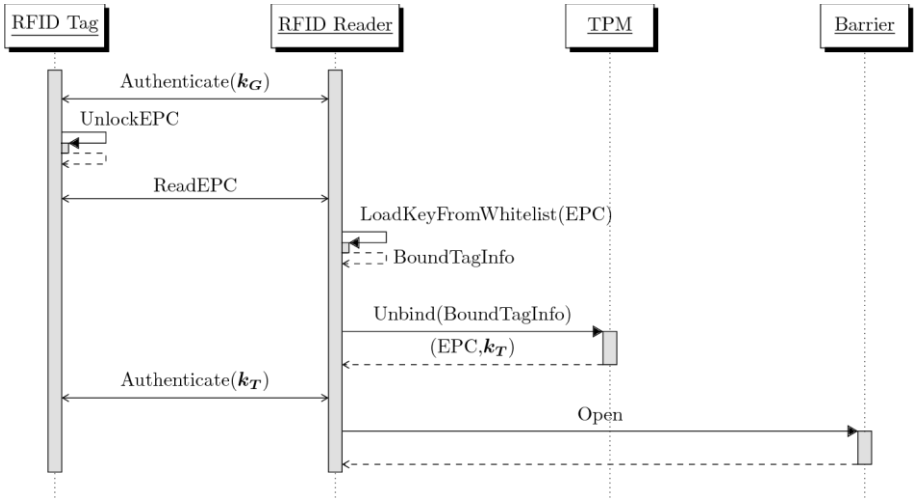

Figure 14: RFID authentication procedure

The risk assessment for this method is shown in Figure 15. Again, the smartphone is a weak link, however, out of the scope of iKoPA. Furthermore, as the NFC Tag contains a copy of k_T, it could be captured and abused by an attacker and is the responsibility of the user to be kept safe. Attacking the RFID Tag itself to extract a secret key can be assumed infeasible under the assumed attack model, as these chips usually undergo rigorous testing procedures. Relaying the RFID communication is also out of the scope of the attack model. This leaves the attack of the firmware of the barrier (or RFID reader), which should be protected by strong integrity measures. TPM technology can further improve this aspect, with the policy enforcement methods described in section 2.2. The decryption key d_R could be stored by the TPM and bound by a policy, which would prevent its use if the firmware were modified. This way, a compromised RFID reader or barrier would not be able to unpack the secret key of the RFID tag.

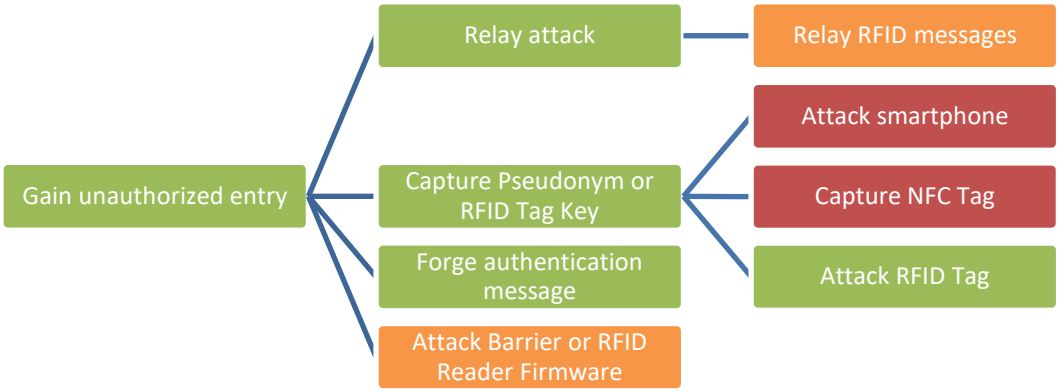

Figure 15: Attack tree for the RFID authentication

3.2.7 Requesting a parked vehicle

Requesting the vehicle must only be possible for authorized users, i.e. the drivers in possession of the car key. As the car is requested using a smartphone, a chain of trust must be established. A prerequisite for this is a pairing of the smartphone to the vehicle, e.g.

via the WiFi of the vehicle. This must only be possible, if the user is an authorized driver, proven for example, by requiring the ignition of the vehicle to be active. The chain of trust can then be built with the help of the registration service. A possible procedure is shown in Figure 16. Similar to the pseudonym certificate shown in section 3.2.4.1, the vehicle will request a certificate from the registration service. This certificate, however, is an intermediate certificate capable of certifying other keys. The vehicle will then use this in turn to create a certificate for the smartphone for requesting the vehicle, the so-called vehicle request certificate.

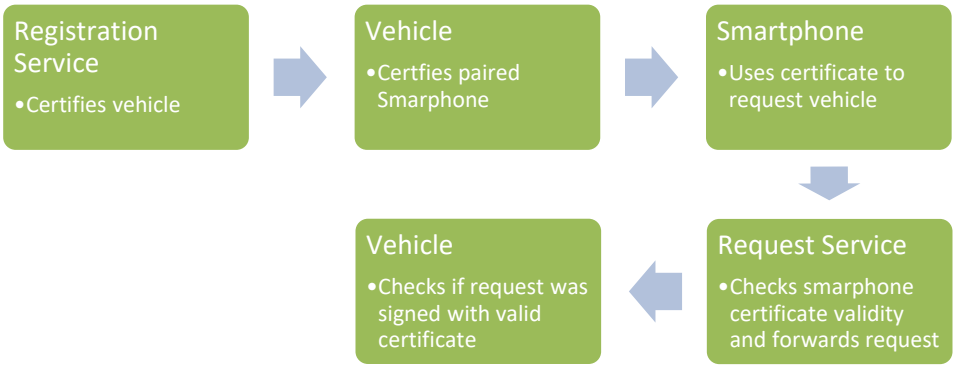

Figure 16: Chain of trust established for the process of requesting a vehicle

This certification path is established beforehand, e.g. shortly before parking the vehicle. The smartphone can then use the vehicle request certificate to request the vehicle. A direct connection between the vehicle and the smartphone for the purpose of a challenge response protocol would be preferable. This may, however, not be possible and a third party such as a request service is necessary, which will forward the request to the car. In this case, the smartphone will use the vehicle request certificate to create an authenticated connection to the request service.

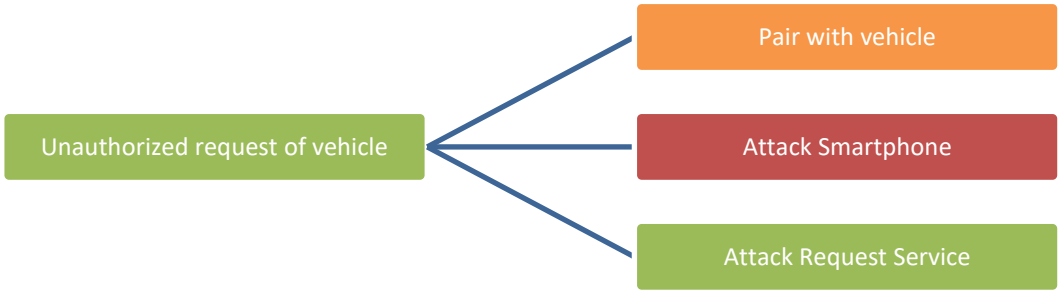

Figure 17: Attack tree for vehicle request

The risk assessment for this scenario is shown in the attack tree in figure 17. As the smartphone has a similar function to a car key in this case, it poses a considerable risk. If an attacker gains control of it, either by software or by theft, an unauthorized request can

be made. This is outside of the scope of iKoPA. The next risk is that of an unauthorized coupling with the vehicle, which would enable an attacker to create a vehicle request certificate. Finally, an attacker might compromise the request service itself. While this is outside of the scope of the attacker model, further steps should be taken. The smartphone could, for example, include a token in the request that is signed with vehicle request certificate. The vehicle is then able to check the tokens signature to attain proof, that the smartphone was active during the request. Possible choices of a token may be a timestamp of the current time, or a random challenge exchanged during the coupling.

3.2.8 Receiving state of charge information

The charge of a vehicles battery can be considered privacy sensitive information, as one might derive the distance driven by the electric vehicle from the state of charge and – with enough samples – deduce the residence location of the driver. The information of the charge should thus be transmitted encrypted to the smartphone. A similar chain of trust as described in section 3.2.7 should be used. However, instead of creating a keypair for authorizing a request, the smartphone will create a keypair for asymmetric encryption. The vehicle can then encrypt and send the state of charge to the smartphone.

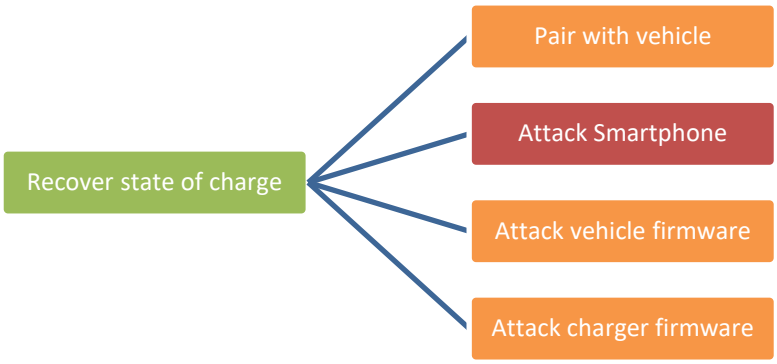

Figure 18: Attack tree for recovering state of charge

Similar to the case shown in section 3.2.7, the weakest point is that of the smartphone, which is outside of the scope of iKoPA. The coupling between smartphone and vehicle must be protected similarly to section 3.2.7. Further risks are attacks on the firmware of either the charger or the vehicle itself. An attacker might leak the state of charge, if a malicious firmware can be executed. Integrity protection mechanisms should be employed accordingly.

4 SCALABLE IT-SECURITY PLATFORM

In this section, we describe a Scalable IT-Security Platform (SISP), which is capable of running multiple independent applications on a single platform, each in isolated execution environments. This enables the consolidation of multiple independent applications or services on a single hardware platform, as well as the separation of services critical to the security of the system. This section is structured as follows: section 4.1 gives a rationale for a choice of an isolation technique that will be used for SISP. Section 4.2 details, how SISP will perform integrity measurements. Section 4.3 describes the general architecture of SISP. Section 4.4 introduces a possible application of SISP in the context of iKoPA.

4.1 Choice of Isolation Technique

The three isolation techniques described in section 2.3 were considered: microkernels, virtualization (hypervisors) and OS-Level virtualization. As one of the goals of SISP is the ability to deploy it on a wide range of hardware, the minimal requirements for the target hardware need to be set. The weakest possible hardware platform we expect SISP to run on is a small processor with a clock frequency below 1GHz, with less than 512MB of memory available. A popular example for such a platform is the "RaspberryPi"[11].

Using a hypervisor for virtualization was rejected due to large resource overhead and hardware requirements. The expected resource consumption is too high for platforms as described above. Additionally, a hypervisor requires specialized hardware support to run at acceptable speeds. This may not be available for all platforms. Mikrokernels were rejected on the grounds of insufficient platform support and potential development effort for application developers. While the situation in terms of resource requirements is better than that of full virtualization, the support for a wide range of platforms is lacking. Furthermore, the effort of developing applications for microkernels is potentially higher than developing for ordinary systems. Projects such as "L4Linux"[12] allow developers to deploy a Linux system on top of a microkernel, however, this would negate the advantage in terms of resource requirements compared to virtualization.

The solution that will be pursued for SISP is that of the OS-Level Virtualization. A Linux based solution was chosen due to its virtually universal platform support. The approach is lightweight enough for smaller platforms and provides a suffient degree of isolation (see [17]). The specific solution is "systemd-nspawn", a component of "systemd"[13], which allows the creation and execution of isolated environments.

[11] https://www.raspberrypi.org/

[12] https://l4linux.org

[13] https://www.freedesktop.org/wiki/Software/systemd/

4.2 Integrity Measurement

As described in section 2.2.1, a TPM requires the assistance of the software running on the system to perform a measurement. A simple approach would be to perform a single large measurement by calculating the hash of the complete storage and extending a PCR value of the TPM, before starting any applications. This may pose a problem in the case of larger applications, as this may block the application from starting for a significant amount of time. Another approach is the measurement of files, as they are read from storage, implemented by the Linux Integrity Measurement Architecture (IMA) [24]. This, however, suffers from a few drawbacks. The measurements must always occur in the same order to produce the same PCR value, which might not be the case when multiprocessor systems are used. Additionally, an attacker might change the files after a measurement was taken. Another problem of IMA is the management of whitelists, which contain well-known file measurements for all necessary files. The necessity to maintain these lists for many possible and changing system configurations creates a hardly manageable operational overhead.

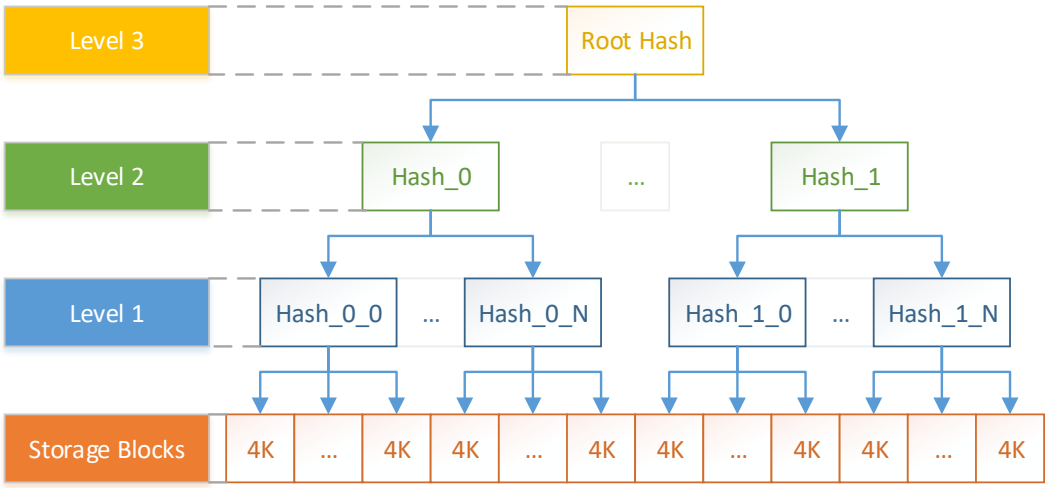

Figure 19: Hash tree structure used for integrity measurement

For SISP an approach similar to that described in [25] is chosen. This approach ensures the integrity of the entire storage; however, instead of measuring the entire storage at the time of boot, the measurement is performed ahead of time and stored in a “hash tree” structure, as shown in figure 19. The storage is divided in blocks of equal size, each of which will be measured and a hash will be stored. In multiple levels, these hashes will then again be hashed and a tree structure is built. The root of the tree is then a hash, which represents a measurement of the entire storage. At the time of boot, the root hash is then

simply read and taken as a measurement of the storage. The "dm-verity"[14] functionality of the Linux kernel then measures each block of the storage as it is read, comparing the measurement to the expected result given by the hash tree.

4.3 General Architecture

Figure 20 shows the architecture for SISP. The Linux kernel runs directly on top of the hardware and provides the framework for the application isolation. During boot, the file systems for the different applications are mounted with the integrity measures described above in place. As an additional measure for saving resources, the application may share one file system as a basis, other application specific file systems will be mounted on top of the base, for example, using the "overlayfs"[15] mechanism provided by Linux. After all application specific file systems are prepared and the integrity measurements were taken, the isolation is set up and the applications are started. Depending on the application, either a complete Linux environment may be started within the isolation if required by the application, or a single process is isolated as a very lightweight option.

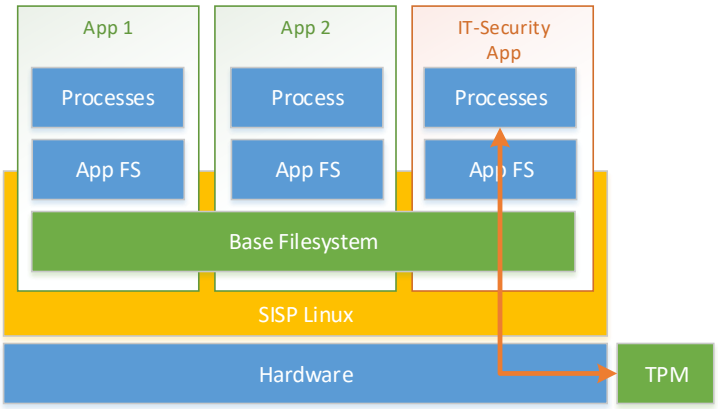

Figure 20: The SISP architecture

4.4 Example Application

An example for deploying SISP is shown in figure 21, in the context of the use case described in section 3.2.6. Due to their sensitive nature, the cryptographic keys involved in the RFID authentication protocol must be strongly protected. One conceivable SISP setup for achieving this protection is the separation of the RFID Reader logic and the key management. The key management handles the cryptographic keys, e.g. receiving and de-

[14] https://www.kernel.org/doc/Documentation/device-mapper/verity.txt

[15] https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

crypting them. As the RFID Reader logic must interact with the key management to perform the authentication protocol, the two isolated compartments are interconnected with an internal network.

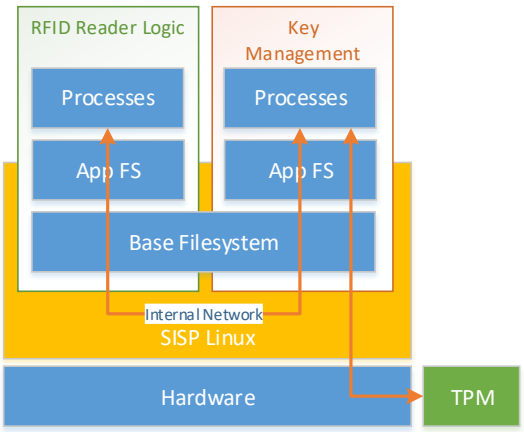

Figure 21: Using SISP for securing the RFID Reader

4.4.1 Distribution and management of keys

One major challenge of such a protection mechanism is the secure distribution and storage of keys while maintaining the privacy of users. The case of the RFID Reader application described above poses additional problems: common RFID based authentication mechanisms only support authentication protocols based on symmetric cryptography. Due to this limitation, cryptographic keys must be handled with particular care, as credentials using symmetric keys are always transferable, i.e. an attacker with knowledge of the key can impersonate a particular user. Furthermore, keys are usually unique to a user (or RFID tag), thus enabling identification of a user.

The RFID tags used in iKoPA offer a privacy protection functionality (cf. D1v2). Essentially, each tag can store and use two separate symmetric AES keys: one (TAM2) intended for protecting the unique EPC identifier, another (TAM1) to authenticate the tag. In practice, TAM2 is a key shared between a group of tags, while TAM1 is unique to each tag. Thus, knowledge of TAM2 enables a RFID Reader to read the protected EPC identifier, i.e. the identification of a tag. Subsequently, TAM1 is then used to authenticate an identified tag. Hence, the privacy of a tag hinges on the protection and safe storage of TAM2, while the security of the application (e.g. access control before a barrier), hinges on the protection of TAM1 (see section 3.2.6).

In the application example described here, an effective protection can be implemented using a TPM. In the case of TAM2, this is a classic use case for a TPM called *sealing*. During deployment, the TAM2 key is sealed, i.e. stored in encrypted form within TPM data-structure, using the TPM2_Create functionality. Ideally, the key object is created with some policy. A classic example is the TPM2_PolicyPCR functionality, which will enforce that a key can only be unsealed if certain PCR values are present during unsealing. This in turn will enforce, that the sealed key can only be used if a specific software is used by the RFID

Reader. Since the functionality of TPM2 has been greatly extended (cf. section 2.2.2), the type of policy can be complex. Since the TAM2 is identical to all RFID tags (or a group of RFID tags), it will be sealed during the initial installation and deployment of the RFID reader.

For TAM1, classic key sealing is not a realistic approach to key distribution. Since TAM1 is unique to each tag, as the number of tags could be arbitrarily large and constantly changing as new tags are introduced. A possible solution, as described in section 3.2.6, is the concept of *binding*. In this case, an asymmetric key-pair is created, for example, using RSA. This key should ideally be unique to each RFID reader. Similar to the case of sealing, this key-pair can be bound to some policy. The public key of this pair is then distributed and can then be used to encrypt data, which then can only be decrypted by the TPM, which created the binding key. In the case of the RFID reader application example, the owners of the RFID tag will encrypt the TAM1 using the binding key and transmit it to the RFID reader ahead of time. In the context of iKoPA, this happens during reservation. The RFID reader then uses the TPM to decrypt the TAM1 to use during authentication.

Protecting the keys with a TPM enforced policy thwarts attack scenarios that include a RFID reader with manipulated software. A manipulated reader could, for instance, be used to extract the keys for identity theft (copying RFID tags) or simply the tracking of users.

4.4.2 Separation of key handling

While the key distribution and management approach described in section 4.4.1 can effectively protect key material during transmission and storage, the protection during *use* of the keys can be further enhanced using the isolation techniques described in section 4.1. To that end, a key management service was defined, which manages the storage of said keys and offers indirect access to the stored keys. As the RFID application does not require actual access to any keys, but rather an access to the en- and decryption functions using these keys, which may then be “hidden” behind a simple key handle. The key management service simply exposes an interface that allows the following operations:

- A lookup function, which checks whether a key with a specific handle is present.
- An encryption function, which performs the AES encryption operation on a given plaintext for a key with a given handle.
- A matching decryption function.

These functions are exposed with a remote procedure call (RPC) interface by a key management server. However, instead of exposing this interface over a common (IP-based) network, UNIX domain sockets are used to allow very tight control over which other applications may have access to the key management.

Using the “nspawn” isolation mechanism described in section 4.1, the RFID application and key management service application are each executed within separate containers. These containers each consist of a file system containing a basic Linux system with all the

prerequisites for the application as well as the application itself. The “nspawn” mechanism can then start this Linux file system within a contained environment. Interfaces between these containers can be tightly controlled, for example, whether a network exists between containers or whether a file system outside of a container is visible to it (file system overlaying). In the case of the RFID reader application described here, the UNIX domain socket of the key management service running in one container is made available to the RFID application running in the other container. No other type of communication is made available, to reduce the attack surface to the absolute minimum. The RFID application container, however, is connected to the network of the actual hardware, as to allow it to communicate with the necessary RFID reader hardware.

To build such application file systems for containers, any development tool capable of producing bootable Linux systems can be used, for example, the installation tools of many Linux distributions. Here, the “mkosi” [26] utility was used, as it is capable of creating file systems protected by the “dm-verity” mechanism described in section 4.2.

5 WORKS CITED

[1] iKoPA Project, "Deliverable D1," 2017.

[2] A. J. Menezes, S. A. Vanstone and P. C. V. Oorschot, Handbook of Applied Cryptography, 1 ed., Boca Raton, FL, USA: CRC Press, Inc., 1996.

[3] M. Stevens, E. Bursztein, P. Karpman, A. Albertini and Y. Markov, "The first collision for full SHA-1," 2017. [Online]. Available: https://shattered.io/static/shattered.pdf.

[4] NIST, "FIPS PUB 180-4: Secure Hash Standard," 2015.

[5] NIST, "FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015.

[6] J. Daemen and V. Rijmen, The Design of Rijndael: AES – The Advanced Encryption Standard, 2002.

[7] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," 2015.

[8] K. Moriarty, B. Kaliski, J. Jonsson and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," 2016.

[9] A. Greenberg, "Radio Attack Lets Hackers Steal 24 Different Car Models," 2016. [Online]. Available: https://www.wired.com/2016/03/study-finds-24-car-models-open-unlocking-ignition-hack/.

[10] K. Wang, s. Chen and A. Pan, "Time and Position Spoofing with Open Source Projects," in *Blackhat*, 2015.

[11] Trusted Computing Group, *TPM Library Specification,* 2014.

[12] W. Arthur, D. Challener and W. K. Goldman, A Practical Guide to TPM 2.0, Apress, 2015.

[13] A. Fuchs, C. Krauß and J. Repp, "Advanced Remote Firmware Upgrades Using TPM 2.0," in *ICT Systems Security and Privacy Protection: 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30 - June 1, 2016, Proceedings*, Cham, 2016.

[14] K. lphinstone and G. Heiser, "From L3 to seL4 – what have we learnt in 20 years of L4 microkernels?," in *ACM Symposium on Operating Systems Principles*, Farmington, PA, USA, 2013.

[15] D. a. B. R. Potts, L. Andresen, J. Andronick, G. Klein and G. Heiser, "Mathematically Verified Software Kernels: Raising the Bar for High Assurance Implementations," 2014.

[16] BSI, "IT-Grundschutz-Kataloge," 2016.

[17] E. Reshetova, J. Karhunen, T. Nyman and N. Asokan, "Security of OS-Level Virtualization Technologies," in *Secure IT Systems. NordSec 2014.*, Cham, 2014.

[18] P. Kamp and R. N. M. Watson, "Jails: Confining the omnipotent root," in *In Proc. 2nd Intl. SANE Conference*, Maastricht, 2000.

[19] G. Terhorst, "Automatisierte Sicherheitstests sicherheitskritischer DAB-Receiver," 2016.

[20] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," 2008.

[21] M. Nystrom and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7," 2000.

[22] Google Inc., "Android Keystore System," [Online]. Available: https://developer.android.com/training/articles/keystore.html.

[23] P. Stavroulakis and M. Stamp, Handbook of Information and Communication Security, Springer, 2010.

[24] J. Corbet, "The Integrity Measurement Architecture," 2005. [Online]. Available: https://lwn.net/Articles/137306/.

[25] Google Inc., "Verified Boot," 2017. [Online]. Available: https://source.android.com/security/verifiedboot/.

[26] J. Liedtke, J. Bartling, U. Beyer, D. Heinrichs, R. Ruland and G. Szalay, "Two years of experience with a microKernel based OS," *SIGOPS Oper. Syst. Rev.,* vol. 25, pp. 51-62, 1991.