A Security Architecture for Multipurpose ECUs in Vehicles

Frederic Stumpf, Fraunhofer-Institute SIT, München
Christian Meves, BMW Group Research and Technology, München
Benjamin Weyl, BMW Group Research and Technology, München
Marko Wolf, escrypt GmbH, München

Kurzfassung

Dieser Beitrag stellt Konzepte und Mechanismen zur Absicherung multifunktionaler Steuergeräte in Fahrzeugen vor. Insbesondere gilt es den Einsatz von vertrauenswürdigen, sicherheitskritischen Anwendungen neben Anwendungen mit einer geringeren Vertrauenswürdigkeit auf einer Plattform zu ermöglichen, ohne die Sicherheit der vertrauenswürdigen Anwendungen zu gefährden. Dazu wird in diesem Beitrag eine Sicherheitsarchitektur vorgestellt, die Anwendungen wirkungsvoll voneinander abschottet und einen sicheren Kommunikationsfluss gewährleistet. Die Architektur stellt weiterhin sicher, dass sicherheitskritische Anwendungen ausschließlich in vertrauenswürdigen Umgebungen voll funktionsfähig sind. Das bedeutet, dass Manipulationen an der Fahrzeugsoftware erkennbar sein müssen und nur Anwendungen in einem akzeptierten Zustand den Zugriff auf weitere Fahrzeugkomponenten erhalten dürfen.

1. Introduction

Current research activities in vehicular on-board IT architectures basically follow two key trends: unification of network communication and centralization of functionality.

Recent on-board IT architectures comprise a very heterogeneous landscape of communication network technologies, e.g., CAN, LIN, FlexRay and MOST. Internet Protocol (IP) based communication is currently being researched as a technology for unifying the overall interconnection of ECUs in future on-board communication systems [14].

State-of-the art vehicular on-board architectures consist of up to 70 Electronic Control Units (ECU), which are interconnected via different bus communication systems. In order to substantiate economical aspects, such as vehicular weight reduction by saving ECU hardware resources or increasing maintainability, the trend is towards centralization of functionality. The centralization of single ECU functions on multipurpose ECUs allows for significant reduction of the overall number of in-vehicle ECUs and hence helps to mitigate the

overall complexity [20]. Open standards and interfaces need to be created in order to create platforms, where software from different suppliers can be steadily integrated [19].

Besides these trends in the design of automotive on-board IT architectures, new external communication interfaces, fixed and wireless are becoming an integral part of on-board architectures. One key factor for this development is the integration of future e-Safety applications based on V2X[1] communications [10,12] that have been identified as one promising measure for decreasing the number of fatal traffic accidents. In addition, user electronic devices, such as smartphones or music players are increasingly integrated in in-vehicle infotainment systems. However, adding external interfaces to vehicular on-board architectures poses new security threats to these systems. Specifically safety applications need to be secured against malicious attacks [11].

Key security requirements of safety applications and safety related information are integrity, authenticity, and trustworthiness. Applying virtualization techniques on multipurpose ECUs support these security requirements, as this technology provides the separation of processes and applications, and hence allows for the execution of applications with different trust levels on the same ECU. As application domains with varying trust levels may communicate within the on-board network, it is reasonable to additionally provide respective security measures in order to control the information flow amongst such domains. We believe that the combination of virtualization and Trusted Computing technologies (e.g., based on Trusted Platform Modules [9]) provides the measures for meeting our discussed security requirements.

An approach to provide a software and hardware security architecture will be the outcome of the EVITA project (*E-safety Vehicle Intrusion proTected Applications*) of the Seventh European Framework Program, which aims to design, verify, and prototype a security architecture for vehicular on-board networks, where all security-relevant components and sensitive information are protected against tampering and malicious manipulations [13]. Thus, EVITA implements the security technology for vehicular communication endpoints, enabling the security of safety applications, but also most other V2X communication applications (e.g., vehicular comfort, or business applications).

---

[1] V2X is an abbreviation stands for any external vehicular communications such as Vehicle-to-Vehicle (V2V) or Vehicle-to-Infrastructure (V2I) communications.

1.1 Our Contribution

This paper specifically focuses on security aspects of communication between multipurpose ECUs based on virtualization technology, where applications can be securely isolated from each other. We propose a security architecture for multipurpose ECUs that is based on virtualization techniques and Trusted Computing technology. Instead of equipping each ECU running on a multipurpose ECU with additional security services, we introduce a dedicated *security controller*. This approach requires the ECUs to only implement minimal security services and, thus, easy migration of existing ECUs to our security architecture. The security controller operates in an isolated domain and is in charge of controlling the information flow between applications. Our security architecture enables communication within the on-board network as well as with external entities in a trustworthy manner by evaluating the integrity and trustworthiness of a sender. For this purpose, we present a security protocol used by the multipurpose ECUs in order to securely exchange data. We finally have implemented parts of our approach in a proof-of-concept prototype to ensure that our approach is feasible.

1.2 Outline

The remainder of this paper is organized as follows. In Section 2, we present background information about virtualization techniques and Trusted Computing technology our solution is based upon. Section 3 discusses the overall scenario we are considering and in Section 4, we present the architecture of our multipurpose ECUs. In Section 5, we show how we realize the integrity reporting in our multipurpose ECU architecture including the security protocols for message exchange. In Section 6, we present details about our proof-of-concept prototype. Finally, we conclude with Section 7.

2. Background

In this section, we present some background information about virtualization techniques and Trusted Computing technology.

2.1 Virtualization

Virtualization basically means realizing several runtime environments in parallel but strictly isolated on a shared hardware. Nowadays, virtualization is an accepted standard and is used in the desktop and server market as a genuine alternative solution to several individual dedicated hardware systems. Through today’s availability of modern and highly efficient virtualization solutions, virtualization becomes extremely interesting also for vehicular applications [3].

As depicted in Figure 1, the concept of virtualization is based on an additional abstraction layer, called the Virtual Machine Monitor (VMM) or hypervisor that is situated between the hardware layer and the operating system(s) or application(s). In practice, this abstraction layer can be realized in hardware, in software, or by a hardware/software combination. The main task of the VMM is to enable the sharing of the real physical resources with all existing runtime environments executed in parallel, called Virtual Machines (VM), without causing any resource conflicts or inconsistencies; in one word: to virtualize. The utilization of the virtualized hardware resources, in turn, has to be transparent for each VM in a way that it can be executed in almost the same manner as a single individual process on its dedicated hardware. The mutual strict isolation, the access control to all shared hardware resources and the control of the VMs itself is managed by the VMM. That means, the VMM implements all effective access policies for all communications, applications, and data as well as for all shared hardware resources. Therefore, the VMM is the actual crucial component in all virtualization concepts for realizing and enforcing the operational IT safety and the IT security as well.

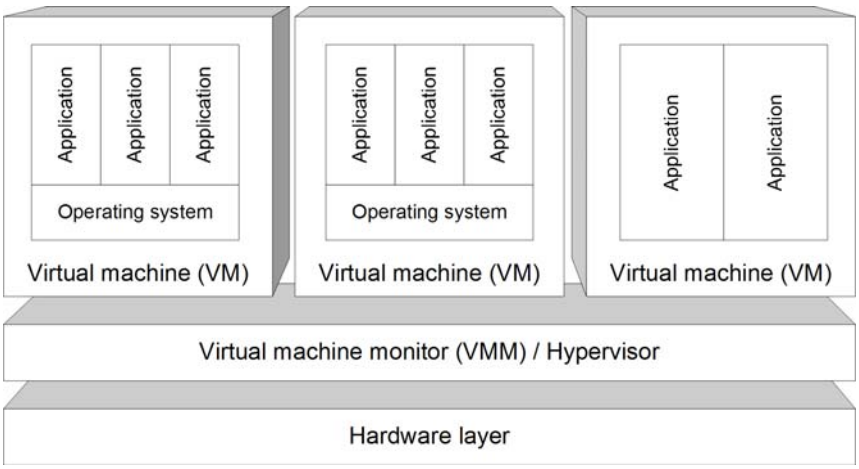

Figure 1: Structure of a virtualized IT Architecture

As described in more detail in [3], the application of virtualization technologies for vehicular ECUs enables various benefits and advantages such as reduction of hardware costs, increased hardware efficiency, peak performance, flexibility, and interoperability as well as especially increased IT safety and IT security while enabling a multilevel-security and multilevel-safety per ECU architectures.

2.2 Trusted Computing

Trusted Computing technology as proposed by the Trusted Computing Group (TCG) [9] provides a set of basic security components and functionalities (e.g., isolated encryption) that form the base for a larger set of high-level security functions (e.g., platform integrity attestation) that can be built upon. Together with a secure operating system, Trusted Computing (TC) can be used to build an appropriate basis for security architectures with improved security especially for distributed and embedded applications that are executed also in "hostile environments".

The following subsections introduce those TC functionalities, which are significant for the design of our security architecture for protection of multipurpose ECUs in vehicles, that means, (i) the Trusted Platform Module (TPM), (ii) the authenticated boot process, (iii) the sealing/binding functionality, and (iv) the remote attestation functionality.

2.2.1 Trusted Platform Module

The base of TC technology is the standardized Trusted Platform Module (TPM) that is considered to be a tamper-resistant hardware device similar to a smart-card and is assumed to be securely bound to the computing platform. The TPM is primarily used as a root of trust for integrity measurement and reporting and to secure all critical cryptographic operations (cf. following paragraphs). Current TPMs base on the open specification version 1.2 published by the Trusted Computing Group (TCG) [7,8,9]. The TPM hardware ensures that malicious software cannot compromise any cryptographic secrets since all security-critical operations such as key generations and decryption operations are done “on-chip”, so that secret keys do not have to leave the chip.

2.2.2 Authenticated Boot

During an authenticated boot process as proposed by the TCG, any code that will be executed is “measured” before execution concretely by calculating its cryptographic hash value. TC hardware is responsible for the secure storage and authentic provision of these measurement results. The hierarchical measurement chain (e.g., a hash chain) starts at the Core Root of Trust Measurement (CRTM), which has to be trusted a priori by all involved parties that want to evaluate the derived measurements. The CRTM is a small immutable (verifier) code implemented into the boot ROM or similar that is executed at first during the booting process. The step-by-step measuring and execution of the boot strap (starting at the CRTM) covers all consecutive layers that are part of the Trusted Computing Base (TCB). Upon completion of an authenticated boot process, these measurements reflect the

configuration of the currently running hardware and software environment. TC technology, however, remains passive and hence does not (and cannot) prevent a certain (insecure) computing environment from being executed.

2.2.3 Binding and Sealing

A distinctive feature of TC hardware is the ability to not only use passwords as authorization (e.g., for a decryption operation with a specific TPM protected key), but also the integrity measurements determined during the authenticated boot process as described before. Thus, only a platform running a previously defined software or hardware configuration can be authorized to use a certain key. Moreover, the property that a certain key is "bound" to a platform configuration can be certified by the underlying TC hardware. This certification includes the integrity measurements that authorize a platform to employ the key. A remote party can verify the certificate and validate the embedded integrity measurements against "known good" reference configurations before encrypting data with the certified key. While the TC binding mechanism *binds* data only to a certain hardware/software configuration, the TC sealing mechanism additionally includes also always a linkage to the TPM's unique identity (i.e., the Endorsement Key). Thus, sealing is mainly used for sealed storage, which means, to *seal* data of a device to itself.

2.2.4 Remote Attestation

The TC remote attestation functionality is used to report the actual platform hardware and software configuration to an external remote party. To guarantee integrity and freshness of the platform configuration reporting, the corresponding integrity measurement values and a fresh nonce provided by the remote party are digitally signed with an asymmetric key called Attestation Identity Key (AIK) that is linked to the unique identity of the TPM (i.e., the Endorsement Key) that is under the sole control of the TPM. A trusted third party called Privacy Certification Authority (Privacy CA) is used to guarantee the pseudonymity of the AIKs.

3. Scenarios and Security Requirements

The following section describes the underlying scenario for which our security solution has been designed for and derives the corresponding security requirements that have to be fulfilled.

3.1 Scenarios

We consider two different scenarios that are based on trends of on-board vehicular networks as described in the introduction. Within the first scenario, we presume the co-existence of multipurpose ECUs and common ECUs, whilst within the second scenario we consider an architecture where only multifunctional ECUs are deployed. The first scenario can be seen as a first step of evolution where not all ECUs are ported to multipurpose ECUs, but also self-contained, widely autonomous ECUs still exist in the on-board infrastructure. The next step in evolution could be that all ECUs are ported to multipurpose ECUs.

Figure 2 shows a multipurpose ECU communicating with other common ECUs. The multipurpose ECU consists of a virtualization-supporting hardware platform, such as Intel's Atom processor [18], a hypervisor for virtualizing the underlying hardware, and a number of virtual machines (VM). In addition, we assume an automotive-capable hardware-based trust anchor, such as the security module currently being designed and prototyped within the EVITA project [13]. For this reason we used in this proposal the TPM since it is currently the only available security module. Each VM is strongly isolated from other VMs and executes the software environment of a proprietary ECU. In addition, the hypervisor provides mechanisms such as virtual machine inspection that allows monitoring the ECU VMs. Hence, if it detects a state that is considered unsecure or untrusted, it can reset the VM to a known secure state.

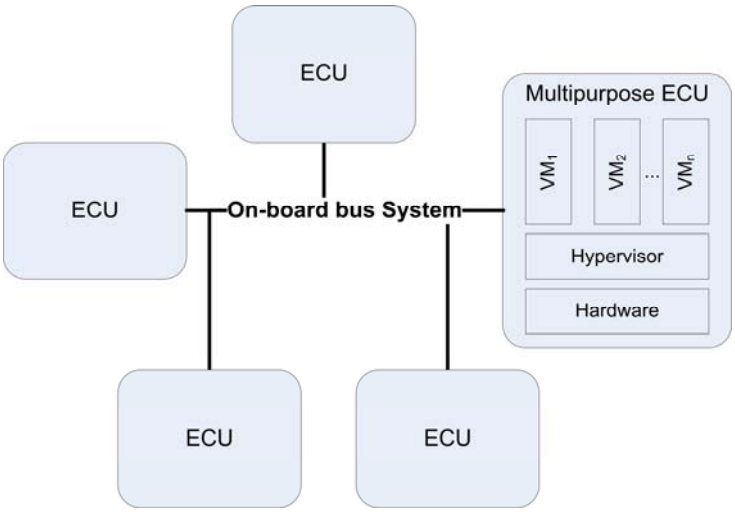

Figure 2: Multipurpose ECU communicating with other ECUs

3.2 Security Requirements

For the scenarios presented above, we define the following security requirements.

- **(SR. 1) Mandatory communication control**: Access control based on authentication, authorization, and integrity and trust verifications can be reliably enforced.
- **(SR. 2) Secure communications**: Confidentiality, integrity, authenticity, and freshness of in-vehicle and external communications can be reliably enforced.
- **(SR. 3) Platform integrity enforcement**: Integrity of in-vehicle ECU platforms can be reliably enforced or modified platform configurations can at least be reliably detected.
- **(SR. 4) Strong runtime isolation**: ECU applications can communicate or access each other's data only over the specified interfaces.

4. Security Architecture

The following section describes the underlying security architecture and the security protocols for the integrity attestation of the ECUs. Figure 3 shows the overall architecture where multiple ECUs (applications) are running on a Virtual Machine Monitor (VMM). The VMM provides an abstraction to the underlying hardware and provides an isolated execution environment for each ECU (satisfying SR. 4). The VMM itself runs on a virtualization-supporting hardware, which is for simplicity reasons not shown in this figure. The figure also shows the three storage locations and the resulting keys that protect access to this storage. The depicted components as well as the secure boot are described in the next subsections.

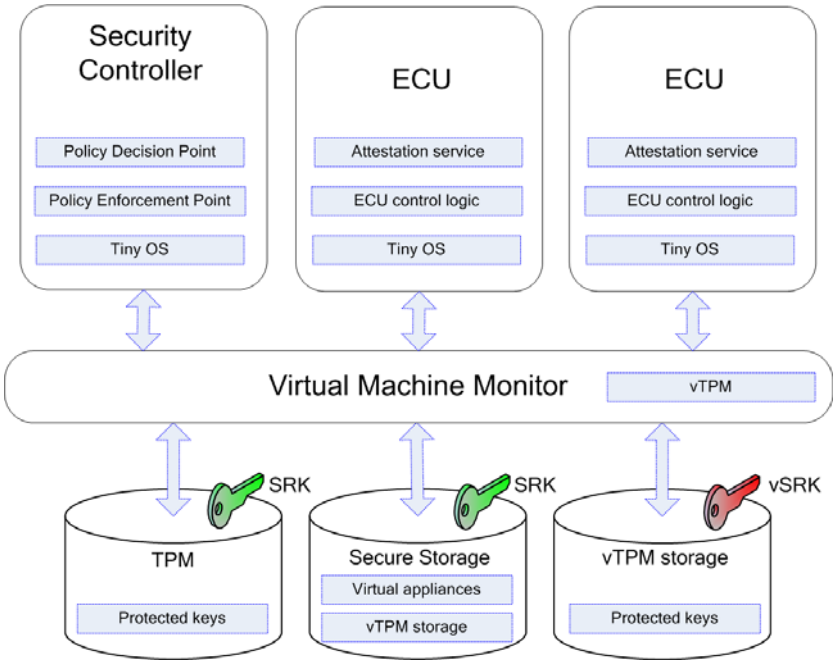

Figure 3: Overall architecture of a multipurpose ECU

4.1 Security Controller

We propose a security controller which is responsible for validating the trust level of a VM and for controlling the communication between VMs and other ECUs. The security controller is a special purpose virtual machine and only one instance of a security controller can run on a multipurpose ECU. All vehicle communication messages are forwarded over the security controller which decides and enforces whether a communication message is forwarded to the destination ECU and whether the message satisfies the required security properties (satisfying SR. 1). A communication message may for example include an integrity proof of the source ECU and can, thus, only be transmitted if this proof is authentic and the ECU is authorized for the requested action. For this purpose, the security controller includes a policy decision point (PDP) where a pre-defined policy specifies which ECUs are allowed to communicate with which ECUs and which security requirements the transmitted messages must satisfy (e.g., secure communication cf. SR. 2). The decision of the PDP is then enforced by the policy enforcement point (PEP) which drops a message, forwards a message or modifies a message (e.g., encrypts the message with the cryptographic key of the destination ECU) according to the policy. Besides deciding whether a message is forwarded or not, the security controller controls the access to the secure storage, where the virtual appliances of the virtual machines are stored. Thus, after the security controller has been executed, it unseals the virtual appliances from the secure storage and executes the virtual machine (see Section 4.2 for additional information).

Formally, the protection state of a multipurpose ECU as protected by the security controller in time t is described using the matrix $\mathcal{M}_t$ which is modeled as follows:

$$\mathcal{M}_t : \mathcal{S}_t \times \mathcal{O}_t \times \mathcal{C}_t \rightarrow 2^{\mathcal{R}}$$

- $\mathcal{S}_t$ denotes the set of subjects whereas $\mathcal{S}_t \subseteq \mathcal{O}_t$.
- $\mathcal{O}_t$ denotes the set of objects.
- $\mathcal{C}_t$ denotes the set of conditions $\mathcal{C}_t = \{(0,0),(1,0),(0,1),(1,1)\}$ where a $(0,0) \in \mathcal{C}_t$ denotes that both $s \in \mathcal{S}$ *and* $o \in \mathcal{O}$ do not provide an integrity proof, i.e., they are not trusted.
- $\mathcal{R}$ denotes the set of rights, for example, $\mathcal{R} = \{send,\ receive\}$.

A matrix entry $\mathcal{M}_t(s,o,c) \subseteq \mathcal{R}$ describes the set of rights that a subject $s \in \mathcal{S}$ has on an object $o \in \mathcal{O}$ at time *t* under a certain condition $c \in \mathcal{C}_t$.

This model is then used to construct the access control matrix $\mathcal{M}_t$, which is a three-dimensional matrix. Examplarily assume that three ECUs (ECU_1, ECU_2, ECU_3) are each located in a separate domain of one multipurpose ECU. ECU_1 measures the distance to the vehicle ahead (e.g., by using radar waves) and transmits a message to ECU_3 if the distance changes (comparable to the active cruise control (ACC) already integrated in modern vehicles). ECU_3 is allowed to interfere with the speed control of the vehicle and can brake or accelerate the vehicle. However, since interfering with the speed control of the vehicle is a safety critical task, the trustworthiness of ECU_1 must be ensured. Thus, ECU_3 only accepts messages if ECU_1 is in a provable and secure state. Vice versa, the driver can adjust the desired distance of the vehicle driving ahead using the Human Machine Interface (HMI) attached to ECU_2. The delivered distance control message is not necessarily security critical and, thus, does not require an integrity proof. Please note that this is a very simple example use case and that even though that many vehicles already have comparable systems, it is not necessarily realized as described here. Table 1 and Table 2 examplary show how parts of the resulting matrix could look like.

		$\mathcal{O}$		
		ECU_1	ECU_2	ECU_3
$\mathcal{S}$	ECU_1			*receive*
	ECU_2	*send*		
	ECU_3			

Table 1: A very simple example policy for $C_t = (0, 0)$ indicating that both subject and object do not provide an integrity proof.

		$\mathcal{O}$		
		ECU_1	ECU_2	ECU_3
$\mathcal{S}$	ECU_1		*receive*	*receive*
	ECU_2	*send*		
	ECU_3	*send*		

Table 2: A very simple example policy for $C_t = (1, 1)$ indicating that both subject and object do provide an integrity proof.

This concept enables the integration of untrusted and trusted applications on one multipurpose ECU. In addition, it prevents untrusted applications from inflicting damage to

trusted applications, e.g., by injecting malicious messages, such as, malicious and non-compliant brake commands.

4.2 Virtual Machines

A virtual machine (VM) represents the interface to the bare hardware constructed by the underlying hypervisor [2]. Each VM runs its own software environment often referred to as *virtual appliances* [4] that consist of a fully pre-installed and pre-configured application and operating system (OS). A virtual appliance is usually configured to host only a single application (the firmware of a particular ECU) and the included OS is adapted (i.e., minimized) to the essential application's need. This approach allows for good efficiency and flexibility since existing and proprietary ECU hardware/software environments can easily be migrated into this new environment.

Each time a new VM is created, a virtual TPM instance is initiated and pre-configured (cf. Section 4.3). The VM also provides an attestation service, which enables accessing the content of the platform configuration register and, thus, the secure reporting of its underlying integrity state to a remote entity.

In order to reduce vulnerabilities, the VM distinguishes between program memory and data memory, similar to the Harvard architecture. The program memory holds the program machine code represented by a sequence of instructions and the data memory holds data that are related to the ECUs or security controller's state. Any modifications to the program memory cannot be written back to the virtual appliance, meaning that each time before the VM spawns, its program memory is reverted to its initial state. Isolating program and data memory are realized through two different disk images. The secondary disk image is used to store state specific data. However, since data stored on the secondary disk image may be able to influence the runtime condition, only data that originates from the VM is stored there.

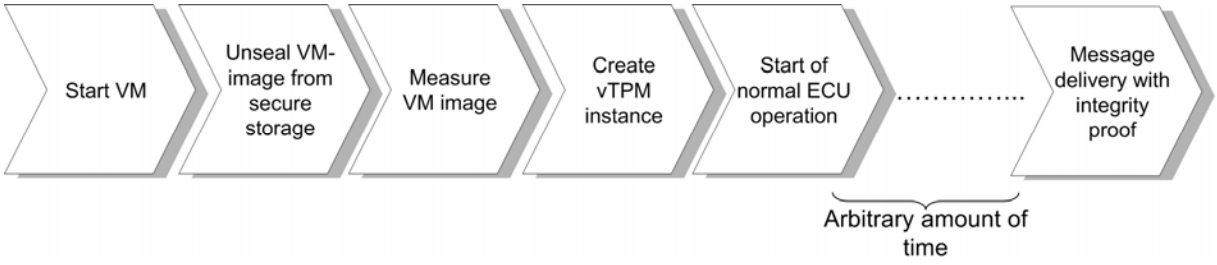

Figure 4: Secure boot mechanism actions taken when a multipurpose ECU spawns

The actions taken when a VM spawns are shown in Figure 4 that includes the unsealing (cf. Section 2.2.3) of the virtual appliance (often also referred to as *image*) and the initialization of

the VM's virtual TPM. Booting a VM is triggered by the security controller which unseals the image of the VM from the secure storage and spawns a VM. Note that the creation of a vTPM instance requires unsealing of the vTPM storage using the SRK. As soon as the VM spawned, it can provide a vTPM signed certificate about its integrity state using the attestation service (cf. Section 5).

4.3 Virtual TPM (vTPM)

An important precondition for placing trust in a remote entity is the establishment of a complete integrity measurement chain from the hardware-based security anchor up to and including the top ECU application. The hardware TPM is virtualized by providing a virtual software TPM to every VM instance. This approach has firstly been introduced by Berger et al. [1] and we employ these concepts here. The advantages of using a virtual TPM in contrast to only using a hardware TPM are twofold:

- First, it enables to generate a proof of the system configuration of the VM in a very small amount of time since the vTPM is a layer of software and, thus, does not possesses the same performance degradations as currently available hardware TPMs [6].
- Second, multiple VMs running on a virtualization layer can attest their individual system configuration to other entities without publishing the configurations of other VMs executed in parallel.

The persistent storage of a virtual TPM is located inside the virtual machine monitor, which in turn is protected by the hardware TPM. Thus, a vTPM-enhanced virtual machine cannot alter the storage of the vTPM. Each time a new VM is created, a virtual TPM instance is initiated with the sealed vTPM storage, and the PCRs 0-15 are filled with the PCR values from the underlying hardware TPM. Additionally, the hash value of the measured virtual appliance of the security controller (PCR16) and the hash value of the virtual appliance of the VM is stored in the virtual TPM (PCR17) as shown in Table 3.

Platform Configuration Register (PCR)	**Content of TPM (Integrity Measurements)**	**Content of vTPM (Integrity Measurements)**
0..7	CRTM and BIOS	CRTM and BIOS
8..15	Bootloader and VMM	Bootloader and VMM
16	Security Controller	Security Controller
17..	Empty	Virtual appliance of the VM

Table 3: Mapping of the PCR values

In order to report the platform configuration of a VM, a strong binding between vTPM and TPM must exist. Otherwise, it would be possible for the vTPM to report PCR values to a remote entity that are different from the ones that were measured by the underlying hardware TPM. To prevent attacks of this type, we make use of a virtual attestation identity credential (vAIK) which is issued and certified by a valid attestation identity key (located inside the hardware TPM). This concept has been proposed in [5] and prevents using malicious or invalid vAIKs. However, usage of this vAIK is only necessary in the certification phase and not in the attestation phase where an ECU delivers messages to another ECU with an integrity proof (Cf. Section 5).

5 Integrity Reporting and Attestation

Our approach is based on a virtual TPM which is a layer of software and, hence, does not provide the same security level as a classic hardware TPM. Thus, the vTPM must be protected against tampering and be protected by the hardware TPM while allowing fast and secure integrity reporting. We achieve this objective by combining a *secure boot* with a concept that we call *integrity stage checks.* Integrity stage checks ensure that access to a particular component located on a specific stage is only possible if all checks that where primarily performed succeed.

5.1 Secure Boot

In addition to an authenticated boot mechanism as described in Section 2.2.2, we assume that integrity references are locally available that represent a valid ECU configuration. The secure boot process is also based on a *security anchor* that has to be trusted a priori by all parties that rely on the secure boot mechanism. As shown in Figure 5, the secure boot pro-

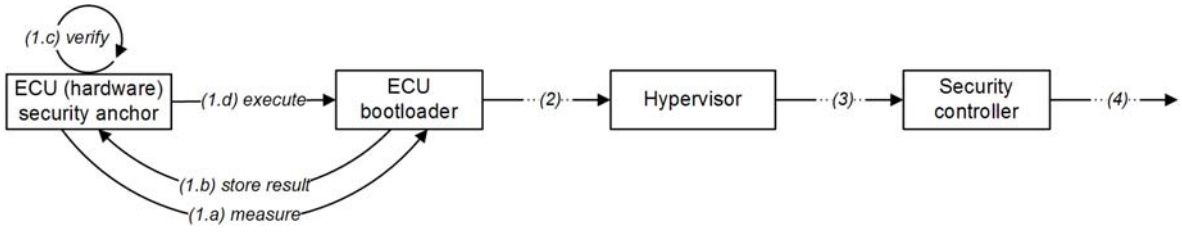

Figure 5: Secure boot chain for integrity protection

cess also memorizes all integrity measurements within the (hardware) protected security anchor for a potential later provision and evaluation. During the depicted secure boot process

- starting at the security anchor - each component (a) first measures (e.g., by calculating its cryptographic hash value) the code of the component that will be executed next while (b) these measurements results are securely stored at the security anchor. The actual measurements then (c) can be verified for correctness by comparing them with the references values securely retrieved either directly from the security anchor or as part of the own configuration that has been successfully verified before. In case of a difference, an alarm can be raised, a pre-defined response can be executed (e.g., boot fail-safe from a ROM) or the boot process can be stopped at all. Finally, the measured and verified consecutive component is (d) executed and takes over control to extend the hierarchical integrity verification chain accordingly. In our case, the secure boot integrity verification consists at least from (1) the security anchor, (2) the bootloader, (3) the hypervisor (later denoted as virtual machine monitor), and (4) the VM bootloader of the security controller.

In order to enable a flexible secure boot (e.g., for security updates), a secure reference update mechanism (e.g., based on a shared secret or a public key scheme) is required, which will not be covered here.

5.2 Integrity Stage Checks

Integrity stage checks ensure that when a specific stage has been successfully passed, the platform satisfies a set of specific security requirements associated to a specific stage, thus, satisfying SR. 3. Integrity stage checks complement the secure boot process described above by extending the secure boot process over the virtualization boundary and by applying more fine-granular checks. The integrity stage checks performed in our architecture are as follows:

Integrity stage check 1: security controller

The first integrity stage check is performed after successful execution of the security controller. For this purpose, the security controller's virtual appliance is sealed to the PCR[0..15] using a key that resides in the hardware TPM, for instance the SRK as shown in Figure 3. The unsealing of the security controller is initiated by the secure boot which measures the VM bootloader and hands over control to the bootloader. The bootloader then unseals the virtual appliance of the security controller and spawns the security controller. As a result, the first integrity stage check can only be passed if PCR[0..15] are in a known state and the virtual appliance of the security controller can be decrypted.

Integrity stage check 2: vTPM-storage

The second integrity stage check can only be passed if the vTPM storage of a specific vTPM instance is in a known and authentic state. For this purpose, the vTPM storage is sealed to the PCR[0..16] of the hardware TPM using the SRK. Note that this approach requires updating and a resealing of the vTPM persistent storage each time new data, such as cryptographic keying material, is placed inside this secure storage of the vTPM.

Integrity stage check 3: ECU specific attestation key

If the preceding integrity stage checks have been successfully passed, the ECU has access to its own associated vTPM. The last integrity stage check is performed before an ECU is able to use his ECU specific key K_{ECU}. This key is bound to the virtual TPM's PCR[0..17] and thus only usable if all previous checks succeed and the vTPM's PCRs are the same as when K_{ECU} was initially bound to. In addition, access to this K_{ECU} is only possible if the security controller is running and, thus, is able to validate all in-vehicle messages originating in an ECU.

5.3 Attestation Protocols

To enable attestation, we divide into an initialization phase and an attestation phase. In this initialization phase, which is typically executed only once, the vTPM is equipped with a special key (K_{ECU}) which is later used for attestation and which is bound to the configuration of the VM. This key is then certified by a trusted party to ensure that the ECU's configuration is known and trusted. The advantage of this concept is that an ECU is able to proof to another entity that it is trusted without requiring the other entity to perform complex computations in order to evaluate the trustworthiness of the ECU.

5.3.1 Initialization Phase

The certification protocol needs only be executed when the software configuration of the ECU changes. Its purpose is to generate a cryptographic key which directly identifies an ECU and is only usable if the software configuration is in the same state as this cryptographic key was initially bound to. The protocol for issuance of such a key and the corresponding certificate is described in the following, where *V* denotes a validator that is able to validate the platform integrity, e.g., the supplier of the ECU firmware, *ECU* is an ECU that wants to receive a certificate, and *SC* is the security controller. First, V must acquire (and validate) the certificate of the Privacy-CA to validate Cert(vAIK, K_{vAIK}). The protocol can be executed over an insecure channel which allows for remotely updating and integrating

new ECU components on a multipurpose hardware platform. The protocol is shown in the following:

ECU Certification Protocol:

---

1. *Protocol messages of the certification phase.*

$$
\begin{aligned}
ECU \rightarrow SC &: Cert(vAIK,\ K_{vAIK}),\ SML,\ \{Cert(ECU,\ K_{ECU}),\ Assertion_{TPM\_PCR\_INFO}\}_{K_{vAIK}^{-1}} && \text{(I)}\\
SC \rightarrow V &: Cert(vAIK,\ K_{vAIK}),\ SML,\ \{Cert(ECU,\ K_{ECU}),\ Assertion_{TPM\_PCR\_INFO}\}_{K_{vAIK}^{-1}} && \text{(II)}\\
V \rightarrow SC &: \{\{timestamp, Certifier, ECU_{prop}, K_{ECU}\}_{K_V^{-1}}\}_{K_{ECU}} && \text{(III)}\\
SC \rightarrow ECU &: \{\{timestamp, Certifier, ECU_{prop}, K_{ECU}\}_{K_V^{-1}}\}_{K_{ECU}} && \text{(IV)}
\end{aligned}
$$

2. *Protocol actions.*

   (a) *Precomputation and Pre-deployment by ECU. ECU* creates a non-migratable vTPM key (K_{ECU}) that is bound to a specific set of platform configuration registers. *ECU* certifies K_{ECU} with K_{vAIK}^{-1}. The resulting structure is denoted $\{Cert(ECU, K_{ECU}), Assertion_{TPM\_PCR\_INFO}\}_{K_{vAIK}^{-1}}$ and includes a `TPM_PCR_INFO` structure which gives an assertion to which *PCR*s K_{ECU} is bound to (compare Table 3). In effect, this structure contains all vTPM PCR values. The resulting certificate is then delivered with the stored measurement log (SML) and the certificate of the *vAIK* in message (I) to *SC*.

   (b) *SC validation.* The security controller validates based on the policy if the ECU is allowed to perform an initialization phase. If this step succeeds, *SC* forwards message (II) to *V*.

   (c) *Integrity validation. V* checks whether the ECU is trusted. For this purpose, *V* processes the *SML* and re-computes the received *PCR* values. If the computed values match the signed *PCR* values, the *SML* is valid and untampered. In addition, the verifiers checks if the used *Cert(vAIK, K_{vAIK})* was derived from a genuine and valid *AIK*. This is done in accordance to the proposed method in [5]. If all steps succeed, *V* creates a certificate $\{timestamp,\ Certifier,\ ECU_{prop},\ K_{ECU}\}_{K_V^{-1}}$ which states that K_{ECU} is bound to a trusted platform configuration. ECU_{prop} is a data structure where all platform characteristics of the ECU, i.e., properties, are encapsulated. This data structure includes information about the type of ECU, its trust level, and other characteristics. This certificate is sent in message (III) to the security controller that forwards the message to the ECU (IV).

5.3.2 Attestation Phase

After the successful initialization phase, the ECU is now in possession of K_{ECU} and the corresponding certificate. The key K_{ECU} can now be used to prove to another ECU that it is trusted, by simply signing a fresh message with this key. The advantage of this concept is that a verifier does only need to verify whether the certificate is valid, rather than parsing the whole measurement chain.

ECU Attestation Protocol:

1. *Protocol messages of the attestation phase.*

$$ECU_{source} \rightarrow SC_{source} : \{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}},\ message,\ \{message\}_{K_{ECU}^{-1}} \quad (1)$$

$$SC_{source} \rightarrow SC_{destination} : \{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}},\ message, \{message\}_{K_{SC_{source}}^{-1}},\ \{message\}_{K_{ECU}^{-1}} \quad (2)$$

$$SC_{destination} \rightarrow ECU_{destination} : message, \{message\}_{K_{SC_{destination}}^{-1}} \quad (3)$$

2. *Protocol actions.*

   (a) *Message creation.* ECU_{source} loads the K_{ECU}^{-1} from the vTPM storage into the vTPM and signs the message that is to be transmitted to $ECU_{destination}$ with K_{ECU}^{-1}. The message includes all information that is necessary to deliver this message to the correct $ECU_{destination}$, i.e., *message* includes information about the source ECU and the destination ECU. The message is then delivered in message (1) together with $\{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}}$ to the source security controller.

   (b) SC_{source} validates based on the policy whether ECU_{source} is allowed to deliver messages to $ECU_{destination}$. If this is true, SC_{source} signs the message with $K_{SC_{source}}^{-1}$ and transmits message (2) to $SC_{destination}$. In addition, this message can also be encrypted with the destination's ECU bound public key to ensure that only an ECU which is in a trusted state can decrypt this message.

   (c) *Integrity validation.* $SC_{destination}$ validates whether the certificates $\{timestamp,\ Certifier,\ ECU_{prop}, K_{ECU}\}_{K_V^{-1}}$ is valid. In addition, $SC_{destination}$ verifies based on the certified platform characteristics (ECU_{prop}) whether ECU_{source} is trusted and whether ECU_{source} is allowed to deliver messages to $SC_{destination}$. If the checks succeed, $SC_{destination}$ signs the message with $K_{SC_{destination}}^{-1}$ and delivers message (3) to the destination ECU.

   (d) $ECU_{destination}$ verifies whether $\{message\}_{K_{SC_{destination}}^{-1}}$ is authentic and processes the message.

6. Implementation

The security controller has been partly prototypically implemented. We have implemented the secure startup of the security controller and access control functionality of the security controller based on policies which can be pre-configured per VM. The prototype uses IP-based communication for communication between VMs. We have integrated the components of the security controller with XEN [17] version 3.2.1 on a standard PC with integrated TPM. Our access control model has been implemented in Java based on the XACML (eXtensible Access Control Markup Language) specification v1.1 [15] and integrated with Squid [16]. Squid intercepts the communication flow and requests a policy decision based on the

XACML standard by the policy decision point, where a permit/deny decision is generated based on the configured policies and the security context. In our prototype, the security context comprises the source and destination IP address and the condition $\mathcal{C}_t$ (i.e., the trust level). The security context is collected by the policy information point and forwarded to the PDP according to the notion of the XACML standard. The described attestation protocol within this paper has yet not been fully implemented.

The security controller image is encrypted with a symmetric key and needs to be decrypted using the corresponding key, which is sealed to the platform. Hence, the key is unsealed first, and then used to decrypt the image. We have performed measurements for the unsealing of the security controller image. First, the private key used for the unsealing process needs to be loaded to the TPM, then, the symmetric key can be unsealed, which is used for decrypting the security controller image. Unsealing this symmetric key on an Intel Core 2 Duo processor takes approx. 2,8 seconds. The decryption of the security controller using the unsealed key is finished after 3,6 seconds. This shows that the architecture is not feasible using a standard TPM and that a security chip with more computation power is required.

7 Conclusion

We have presented a security architecture that supports the deployment of applications with different trust levels on multipurpose ECUs by applying virtualization and Trusted Computing technology. The security architecture comprises a dedicated security controller which is in charge of controlling the communication flow between VMs on the same multipurpose ECU as well as the communication to external entities. This communication can be controlled by specifying a dedicated access control matrix. This matrix considers a set of conditions, e.g., trust for the generation of an access decision (permit/deny). We have proposed a protocol for efficiently issuing and exchanging trust statements within a vehicular on-board network, issued or enforced by a security controller. However, our proposed concept is also applicable for V2X communication scenarios. This security controller operates in a trusted isolated environment and enforces the policies discussed above. It is already feasible to deploy our approach in on-board architectures, where only one multipurpose ECU is available since it enables the secure integration of less trusted applications with trusted environments.

Acknowledgments

This work presents parts of the collaborative project EVITA co-funded by the European Commission under the 7th Framework Program.

References

[1] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: virtualizing the trusted platform module. In USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium, pages 21–21, Berkeley, CA, USA, 2006. USENIX Association.

[2] Robert P. Goldberg. Survey of virtual machine research. Computer, June 1974.

[3] Jan Pelzl, Marko Wolf, and Thomas Wollinger. Virtualization Technologies for Cars: Solutions to increase safety and security of vehicular ECUs. In Automotive – Safety & Security, Stuttgart, Germany, November 19–20, 2008. Shaker Verlag, 2008.

[4] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nickolai Zeldovich, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In Proceedings of the 17th USENIX conference on System administration (LISA '03), pages 181–194, Berkeley, CA, USA, 2003. USENIX Association.

[5] Frederic Stumpf, Michael Benz, Martin Hermanowski, and Claudia Eckert. An Approach to a Trustworthy System Architecture using Virtualization. In Proceedings of the 4th International Conference on Autonomic and Trusted Computing (ATC-2007), volume 4158 of Lecture Notes in Computer Science, pages 191–202, Hong Kong, China, 2007. Springer-Verlag.

[6] Frederic Stumpf, Andreas Fuchs, Stefan Katzenbeisser, and Claudia Eckert. Improving the scalability of platform attestation. In Proceedings of the Third ACM Workshop on Scalable Trusted Computing (ACM STC'08), pages 1–10, Fairfax, USA, October 31 2008. ACM Press.

[7] Trusted Computing Group. TCG PC Client Specific Implementation Specification for Conventional BIOS. Technical report, July 2005.

[8] Trusted Computing Group (TCG). http://www.trustedcomputinggroup.org, 2009

[9] Trusted Computing Group (TCG). Trusted Platform Module (TPM). Main Specification Version 1.2 Revision 103, TPM Work Group, July 2007

[10] Kosch, T.: Local Danger Warning based on Vehicle Ad-hoc Networks: Prototype and Simulation. In Proceedings of *1st International Workshop on Intelligent Transportation (WIT), Hamburg, Germany, March* 2004.

[11] Barisani, A., Bianco, D.: Unusual Car Navigation Tricks. In Proceedings of CanSecWest, Vancouver, Canada, April 2007.

[12] Car-to-Car Communication Consortium (C2C-CC), http://www.car-to-car.org, 2009.

[13] E-safety vehicle intrusion protected applications (EVITA) Project. http://www.evita-project.org, 2009.

[14] Rahmani, M., Hillebrand, J., Hintermaier, W., Bogenberger, R., Steinbach, E.. A Novel Network Architecture for In-Vehicle Audio and Video Communication. In Proceedings of Second IEED/IFIP International Workshop on Broadband Convergence Networks. May 2007.

[15] OASIS. Extensible access control markup language (xacml) version 1.1 committee specification. August 2003.

[16] Squid, http://www.squid-cache.org/, 2009.

[17] Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Pratt, I.; Warfield, A.; Barham, P. & Neugebauer, R. Xen and the Art of Virtualization Proceedings of the ACM Symposium on Operating Systems Principles, 2003.

[18] Product Brief. Intel Atom Processor Z5xx Series for Embedded Computing. http://download.intel.com/design/chipsets/embedded/prodbrf/Atom_Product_Brief.pdf, 2009.

[19] Genivi Alliance, http://www.genivi.org/, 2009.

[20] Holzknecht, S., Biebl, E., Michel, H. Graceful Degradation for Driver Assistance Systems, In Advanced Microsystems for Automotive Applications 2009, pp. 255-265, Springer Verlag, 2009.